VŠB - Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Vývoj aplikací nad platformou Microsoft SharePoint 2010

Application Development on Microsoft SharePoint 2010 Platform

2012 **Michal Duda**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Michal Duda**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Vývoj aplikací nad platformou Microsoft Sharepoint 2010
Application Development on Microsoft Sharepoint 2010 Platform

Zásady pro vypracování:

Zanalyzovat typické požadavky pro nasazení platformy Sharepoint, zmapovat rozdíly v implementaci pro prostředí intranetu, extranetu a internetu. Vytvořit rozšiřující části pro Sharepoint a implementovat ukázkové řešení.

1. Analýza typických požadavků pro nasazení platformy Sharepoint v prostředí intranetu, extranetu a internetu.
2. Systémový design - Hardware, Software požadavky pro nasazení v intranetu, extranetu, internetu.
3. Konfigurace a nasazení.
4. Implementace vlastních rozšíření v systému Sharepoint:
    4.1. přizpůsobení grafického rozhraní, vytvoření vlastních master pages, vyvoření vlastních šablon stránek
    4.2. vytvoření vlastních site definition, rozšíření některých site definition obsažených v systému Sharepoint
    4.3. označování oblíbených blogů a WiKi
    4.4. implementace logiky pro centralizaci dat jednotlivých uživatelů a odpovídajícího grafické rozhraní ve formě web parts
    4.5. implementace web parts umožňujících načítání dat z veřejných zdrojů
    4.6. vytvoření specifického workflow pro schválení a publikování obsahu / dokumentu 5. Implementace typického intranetového řešení na platformě Sharepoint s využitím výše uvedených rozšíření
    5.1. Publikování článků a novinek, vytváření nových publikačních site
    5.2. Vytváření uživatelských blogů, WIKI stránek
    5.3. Vytváření týmových a projektových site

Seznam doporučené odborné literatury:

Joerg Krause, Martin Döring, Christian Langhirt, Bernd Pehlke, Alexander Sterff. SharePoint 2010 as a Development Platform. [s.l.] :
Apress, 2010. 1164 s. ISBN 978-1-4302-2706-9.

MALIK, Sahil. Microsoft SharePoint 2010 : Building Solutions for SharePoint 2010. [s.l.] : Apress, 2010. 400 s. ISBN 978-1-4302-2865-3.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Petr Felzman**

Konzultant diplomové práce: Ing. Martin Milata

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 23. 4. 2012 ..... Michal Dada .....

podpis

Poděkování

Děkuji Mgr. Petrovi Felzmannovi za vedení mé diplomové práce.

Děkuji Bc. Tomáši Gröplovi, odborníkovi na problematiku platformy SharePoint, za mnoho cenných rad.

Děkuji Ing. Martinu Milatovi za konzultace spojené s mou diplomovou prací.

Abstrakt

Tato diplomová práce je vypracována ve spolupráci s firmou Tieto Czech s.r.o. V rámci diplomové práce jsou popsány běžné požadavky kladené na internetová, intranetová a extranetová řešení na platformě SharePoint 2010 a také popis požadavků na hardware a software pro nasazení platformy SharePoint 2010. Hlavním účelem diplomové práce je implementace intranetového řešení. Řešení zahrnuje některé z obvyklých požadavků a bude společnosti Tieto sloužit jako základ pro implementaci dalších zákaznických intranetových aplikací postavených na platformě SharePoint 2010. Řešení bude také sloužit jako referenční implementace pro vývojáře začínající s platformou SharePoint 2010.

Klíčová slova

Microsoft SharePoint 2010, Microsoft SharePoint 2010 Foundation, Microsoft SharePoint 2010 Server, Intranetové řešení, Řešení pro týmovou spolupráci, .NET, C#, Intranet, Unity framework, LINQ to SharePoint

Abstract

Thesis has been made in cooperation with Tieto Czech s.r.o. The thesis describes frequent customer‘s requirements for internet, intranet and extranet SharePoint 2010 solutions as well as hardware and software requirements of SharePoint 2010 platform. Main goal of the thesis is implementation of intranet solution. This solution contains some of common requirements and can be used as root for cusmetr’s projects based on SharePoint 2010. It can be also used as reference implementation for developers who begins with SharePoint 2010 platform.

Key words

Microsoft SharePoint 2010, Microsoft SharePoint 2010 Foundation, Microsoft SharePoint 2010 Server, Intranet solutions, Team collaboration solutions, .NET, C#, Intranet, Unity framework, LINQ to SharePoint

Použité zkratky

XML	eXtensible Markup Language
ASP.NET	Active Server Pages .NET
NLB	Network load balancing
LINQ	Language-Integrated Query
DDF	Data definition file
CAB	Cabinet file
WYSIWYG	What you see is what you get
CSS	Cascading style sheet
CAML	Collaborative Application Markup Language
GAC	Global Assembly Cache
BCS	Business Connectivity Services

Obsah

Úvod

V diplomové práci se zabývám problematikou použití platformy Microsoft SharePoint 2010 pro vývoj podnikových řešení. V rámci diplomové práce jsem implementoval intranetové řešení, které zahrnuje typické rozšiřující požadavky, které bývají řešeny nad platformou SharePoint. Řešení rozšiřuje možnosti platformy SharePoint v oblasti publikačních funkcionalit, blogů, WIKI a týmové spolupráce. Jsou implementována rozšíření jako například kategorizace publikovaných stránek, časovaná úloha pro agregování dat z uživatelem vybraných webů nebo vytváření blogů a WIKI dostupné všem uživatelům intranetu. V diplomové práci rozebírám vytvoření vlastní datové vrsty nad systémem SharePoint, namapování této vrstvy na třídy jazyka C# a další využití pro čtení nebo ukládání dat**.** Také jsem implementoval jednoduchý proces pro publikování stránky pomocí frameworku *Workflow Foundation 3.5* a načítání dat z veřejných zdrojů přes integrační framework systému SharePoint, *Bussiness Connectivity Services*. Většina zákazníků také požaduje změnu výchozího vzhledu systému, zahrnul jsem tedy do své práce i tuto problematiku. Protože je řešení zamýšleno spíše jako framework a referenční implementace pro další rozšiřování, věnoval jsem velkou část návrhu rozšiřitelné několikavrstvé architektuře. Za pomocí Unity frameworku je možné konkrétní implementace tříd jednoduše nahradit novými.

V první části práce se věnuji krátkému teoretickému popisu platformy především pro vysvětlení základních pojmů, popisu nasazování řešení, základní konfiguraci, softwarovým a hardwarovým požadavkům na systém SharePoint a systémové architektuře. Dále také uvádím typické požadavky na intranetová, extranetová a internetová řešení a jejich podchycení v rámci systému SharePoint.

1. Platforma SharePoint 2010

Systém SharePoint je robustním řešením pro správu obsahu, ukládání dokumentů a týmovou spolupráci. Systém bývá používán spíše většími společnostmi. Často je používán pro intranetová a extranetová řešení, případně i internetová řešení. U systému SharePoint se předpokládá nasazení různých komponent systému na rozdílné servery (i když mohou běžet všechny na jednom serveru), čímž je možné vytvořit opravdu robustní podniková řešení.

Na SharePoint je ale potřeba nahlížet i jako na vývojovou platformu. Systém se dá velmi dobře využít jakožto framework s množstvím již hotových funkcí a standardním uživatelským rozhraním. Je proto vhodný pro další rozšíření a vytvoření vlastních podnikových řešení. SharePoint dokonce vytváří vlastní datovou vrstvu nad databází MS SQL Server a přináší tak určité nové možnosti oproti klasické relační databázi.

Systém SharePoint existuje v několika verzích. V nejjednodušší verzi SharePoint Foundation je dokonce systém neplacený, nenabízí však některé pokročilé funkcionality jako například publikační infrastrukturu nebo pokročilé funkcionality pro vyhledávání. Placená verze se nazývá SharePoint Server a je nabízena ve verzi Standart a Enterprise.

1.1. Architektura systému SharePoint

Systém SharePoint Foundation je postaven nad frameworkem ASP.NET verze 3.5 a IIS serverem verze 7. Verze SharePoint Server dále rozšiřuje možnosti verze Foundation [1]. Pro uložení dat je potřeba MS SQL Server, kde jsou ukládána jak data vytvořená uživateli (*content* databáze) tak konfigurační data systému (*config* databáze). Součástí instalace systému je také několik Windows služeb, například služba pro indexování obsahu, služba pro spouštění načasovaných úloh (*Time job*) a podobně.

1.2. Logická struktura systému SharePoint

Logická struktura systému SharePoint má čtyři úrovně. Jsou jimi *farm*, *web application*, *site collection*, *site*.

První úroveň, *farm*, představuje vše, co se týká jedné instalace systému SharePoint, tedy webové aplikace a služby. Z fyzického pohledu se farmou rozumí všechny servery, na kterých systém SharePoint běží.

V rámci jedné farmy může být vytvořena jedna nebo více webových aplikací. Webová aplikace vytvořená v rámci systému SharePoint není nic jiného než ASP.NET webová aplikace, se kterou je možné v IIS serveru pracovat jako s jakoukoliv jinou webovou aplikací [2]. Tedy například nastavit vlastní aplikační *pool* a podobně. Po instalaci je automaticky vytvořena jedna speciální aplikace nazvaná centrální konfigurace pro administraci a konfiguraci systému. K jedné webové aplikaci se také vztahuje jedna nebo více *content* databází**,** ve kterých jsou uložena data spravovaná systémem SharePoint.

Další úrovní je *site collection*. Tedy k jedné webové aplikaci se opět může vztahovat více *site collection*. *Site collection* jsou uložena v *content* databázi. Jedna databáze může obsahovat více *site collection*, ale jedna *site collection* nemůže být rozdělena mezi více databází [2]. *Site collection* představuje kontejner pro *site*, které obvykle sdílí některé společné funkcionality a oprávnění. Ke každé *site collection* existuje právě jedna kořenová *site*.

Poslední úrovní je *site*, neboli také *web site*. **Ve své práci používám název *web site*.** Na této úrovni jsou uloženy seznamy, stránky a dokumenty. *Web site* jsou vytvořeny podle určité šablony, jako jsou například projektové *web site*, blog, WIKI a podobně. [2]

1.3. Datová vrstva systému SharePoint

Data jsou v systému SharePoint ukládána do **seznamů**. Seznamy obsahují sloupce (*fields*) a daly by se takto považovat za podobné klasickým tabulkám z relačních databází. Nicméně určení struktury dat je častěji dáno připojením ***content* typu** než přímo specifikací sloupců. *Content* typ obsahuje sloupce a je tak předpisem struktury, jakou mají mít data uložená jako tento typ. Jeden seznam může podporovat ukládání více než jednoho typu dat a tím se nejvíce odlišuje od klasické tabulky v relační databázi. Sloupce v *content* typech nebo v seznamech mají definován datový typ (*field type*). Soubory a dokumenty jsou ukládány do knihoven, což je speciální případ seznamu. I soubory jsou ukládány pod určitým *content* typem, proto i k souborům bývají uloženy strukturované meta informace. *Content* typy pro dokumenty jsou odvozeny od bázového typu *Document.* Každý záznam uložený v systému SharePoint je spojen s nějakým *content* typem, pokud není dáno jinak tak alespoň s úplně základním *content* typem *Item.* Z tohoto *content* typu jsou odvozeny všechny ostatní.

Systém SharePoint obsahuje množství *content* typů i šablon pro seznamy. Nicméně v rámci vlastních implementací je možné vytvořit vlastní *content* type, šablonu pro seznam i jeho instanci, vlastní definice sloupců i vlastní definice pro typy sloupců.

1.4. Základní pojmy z platformy SharePoint

V popisu intranetového řešení používám pojmy, které jsou notoricky známé všem programátorům pracujícím se systémem SharePoint, avšak pro ostatní by text byl pravděpodobně špatně pochopitelný. Proto uvádím několik základních pojmů a jejich vysvětlení.

Web part

Prvek uživatelského rozhraní. *Web part* rozšiřuje třídu *WebControl*, která je bázovou třídou pro ASP.NET komponenty uživatelského rozhraní. *Web part* je možné dynamicky umísťovat na stránky uložené v databázi systému SharePoint (*Site Page*).

Connectable web parts

Speciální druh *web parts* komponent, které mohou být propojeny a předávat si zprávy. Jedna komponenta obvykle vystupuje v roli poskytovatele dat a druhá v roli příjemce.

Master page

Mají stejný význam jako v klasickém ASP.NET, představují základní *layout* pro více aspx stránek.

Page layout

Rozšiřují dále *master page* a představují předpis pro aspx stránky uložené v *content* databázi systému SharePoint. Využívají se především pro specifikaci rozmístění zón pro *web parts*.

Workflow

V kontextu systému SharePoint pod pojmem *workflow* rozumím automatizovaný business proces pomocí Worflow Foundation frameworku verze 3.5.

Site template

Šablona pro vytvoření *web site*. Může obsahovat specifikaci funkcionalit, které musí být aktivovány při vytvoření *web site* tohoto typu, definici seznamů apod.

Site page

Stránka (aspx soubor) uložená v databází systému. Na tyto stránky mohou být umísťovány *web parts*.

Aplikační stránka

ASP.NET stránka, která není uložena v databázi systému SharePoint, ale v souborovém systému serveru. Na aplikační stránky nemohou být umísťovány *web parts*, nicméně může k ní být připojen serverový kód (*code behind*), což v případě *Site Page* není možné.

LINQ to SharePoint

Od verze SharePoint 2010 je možné pro dotazování nad daty použít objektově orientovaný přístup přes jazyk LINQ. Přes LINQ to SharePoint je možné namapovat seznamy na třídy ve zvoleném programovacím jazyce.

2. Analýza typických požadavků pro nasazení platformy SharePoint

2.1. Prostředí intranetu

Ve společnosti Tieto byly při řešení projektů na platformě SharePoint nalezeny čtyři základní požadavky pro intranetová řešení. Tyto požadavky vycházejí z reálných zkušeností se zákazníky. Těmito čtyřmi základními body jsou:

2.1.1. Personifikace

Naprostá většina zákazníků požaduje, aby uživatelům intranetu byly nabízeny a zobrazovány relevantní informace vzhledem k jejich roli a zájmům v organizaci.

Systém SharePoint umožňuje vytváření skupin uživatelů a nastavování práv těmto skupinám pro objekty na různých logických úrovních v systému. Je tedy snadno možné skrýt před některými uživateli informace, které pro ně nejsou relevantní.

V systému SharePoint je také možné zapnout funkcionalitu MySite, která umožní uživateli vytvoření *web site*, kterou si může sám přizpůsobit pomocí dostupných *web parts*.

2.1.2. Modulárnost

Neboli sestavení intranetu z funkčně nezávislých částí – modulů. Tomuto požadavku vyhovuje systém SharePoint velmi dobře. Naprostá většina funkcionalit je aktivována pomocí tzv. *features*. Ty obsahují rozšiřující prvky, které jsou po aktivaci zavedeny do systému SharePoint. *Features* se používají ať již se jedná o naprogramované rozšiřující řešení nebo funkcionality, které jsou přímo součástí instalace systému SharePoint. Jednotlivé funkcionality mohou být aktivovány na různé logické úrovně v systému (*farm, web application, site collection, web site*).

2.1.3. Distributovatelnost

Distributovatelností je myšlena možnost určovat odpovědnost různých uživatelů pro různé části intranetu. Tuto podmínku systém SharePoint také splňuje díky možnosti vytváření skupin uživatelů a nastavování práv pro různé logické úrovně v systému. Například vedoucí oddělení může být nastaven jako administrátor *site collection* nebo *web site*, která se týká jeho oddělení a být odpovědný za aktuálnost dat.

2.1.4. Agregovatelnost

Agregování dat z různých částí intranetu bývá jedním z nejčastějších rozšiřujících požadavků. Systém SharePoint částečně požadavky pro agregaci relevantních dat splňuje díky MySite, nicméně v mnoha případech je MySite pro uživatele příliš složitá nebo z nějakého důvodu nemůže být tato funkcionalita aktivována. Proto agregace a přehledné zobrazení dat zůstává velmi častým požadavkem pro programátory vytvářející řešení na platformě SharePoint. Také ve svém řešení jsem vytvořil několik funkcionalit pro agregování a zobrazení dat podle aktuálního uživatele.

2.2. Prostředí internetu

Pro správu internetových portálů je samozřejmě zapotřebí funkcionalita pro vytváření obsahu pomocí WYSIWYG editorů. V případě platformy SharePoint je nejlepší možností použití funkcionality nazvané Publishing Site Infrastructure. Ta obsahuje efektivní nástroje pro správu a publikování stránek. Je ale dostupná pouze v placené verzi SharePoint Server. Verze SharePoint Foundation tuto funkcionalitu neobsahuje.

U internetových portálů bývá také obvykle zapotřebí nahradit implicitní grafiku systému podle požadavků zákazníka. Těmto požadavkům není problém vyhovět, v systému SharePoint je možné nahradit výchozí *master* šablony a vytvořit šablony pro publikační stránky.

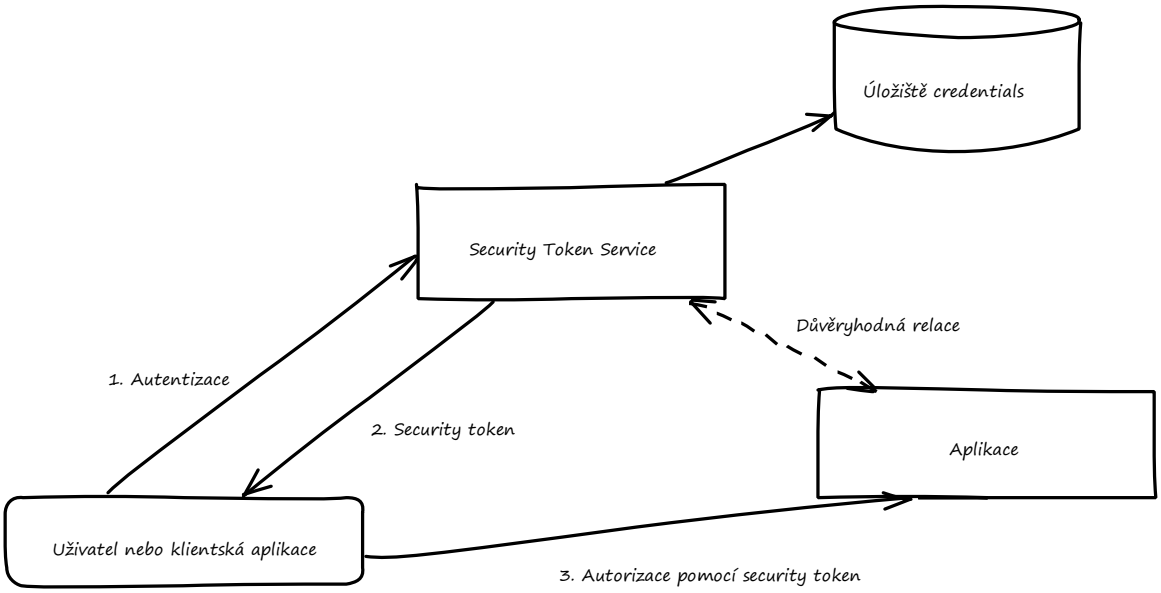
Použití Sharepoitu jakožto platformy pro veřejné internetové stránky naráží ale mnohem častěji na problém s licencováním než na problémy s funkcionalitou, kterou systém nabízí. Licencování systému SharePoint není rozděleno pouze podle nabízené funkcionality, ale také podle počtu uživatelů, kteří se systémem pracují. Pro teoreticky neomezený počet externích uživatelů je zapotřebí licence SharePoint Server 2010 for Internet Sites. Licence musí být zakoupena pro všechny servery, které poskytují služby pro externí uživatele (*front-end* servery, indexovací servery, atd.) [7].

2.3. Prostředí extranetu

Základním požadavkem pro extranetové řešení je možnost vzdáleného přístupu zaměstnanců k interním informacím a aplikacím. Extranet ale nemusí sloužit pouze zaměstnancům pracujícím na dálku, často bývá zapotřebí přístup také pro partnery nebo prodejce. Partnery se rozumí externí spolupracovníci, například z jiných firem. V tomto případě již samozřejmě nechceme, aby měli partneři přístup ke všem korporátním informacím, ale pouze ke zdrojům, které potřebují pro svou spolupráci. V případě prodejců se již bude jednat spíše jen o určitý cílený obsah.

Přístup do extranetu samozřejmě musí být podmíněn autentizací a přístup k jednotlivým informacím autorizací. Samotný systém SharePoint uživatele neautentizuje, využívá určitého poskytovatele autentizace. Ve verzi 2010 byl představen nový způsob autentizace založené na „známkách“ (*tokens*) nazvaný *Claims Based* autentizace. Tento způsob je založen na určité službě (bývá označována jako STS – *Security Token Service*), u které se uživatel autentizuje a získá „známku“, která je potvrzením jeho identity. Mezi aplikací (v tomto případě SharePoint) a STS službou existuje důvěryhodná relace a uživateli je tak na základě „známky“ umožněn přístup do aplikace. Princip bude jasnější z následujícího obrázku (Obrázek 1).

V systému SharePoint 2010 je tedy možné využít více způsobů autentizace uživatele. Uživatelé přicházející z interní sítě mohou být například identifikováni pomocí Windows autentizace, zatímco pro externí uživatele může být využita *Form Based* autentizace. Pro tu je potřeba definovat *membership provider,* kterým může být například klasický *ASP.NET DB membership provider* nebo vlastní naprogramovaný *membership provider*.



Obrázek 1 *Claims Based* autentizace [3]

3. Systémový design

3.1. Hardwarové požadavky

Systém SharePoint klade poměrně vysoké nároky na hardwarovou konfiguraci jak serverů, tak stanic vývojářů, kteří implementují rozšiřující řešení. Následující informace jsou převzaty z oficiálních doporučení společnosti Microsoft, která je možné najít v online *Technet Library* [8].

3.1.1. Web (*front-end*) servery, aplikační servery, *single server installations*

Pojmem *single server installation* se rozumí instalace systému, kdy všechny služby běží na jednom serveru společně i s databázovým serverem.

Pro tyto druhy serverů je nutný 64 bitový procesor, doporučen je čtyřjádrový. 4 GB RAM je doporučeno pro vývojové nebo výukové prostředí a 8 GB RAM pro produkční prostředí. Z vlastní zkušenosti ale vím, že pro seriózní vývoj aplikací 4 GB RAM nestačí, je potřeba také 8 GB. Pro pevný disk je doporučena kapacita alespoň 80 GB.

3.1.2. Databázový server

Pro databázový server je doporučená hardwarová konfigurace 64 bitový čtyř jádrový procesor a 8 GB RAM, případně osmijádrový procesor a 16 GB RAM pro rozsáhlejší řešení. Požadavky na diskovou kapacitu se samozřejmě odvíjí od předpokládaného množství dat, které budou v systému uložena.

V knize *Professional SharePoint 2010 Administration* [4] se tato doporučení dále rozvíjejí, nicméně doporučení jsou stále velmi obecná. 8 GB paměti požadují za naprosté minimum a pro silně zatížený SQL Server požadují 32 GB. Stejně tak čtyřjádrový procesor považují za minimum a za častější konfiguraci považují server s více čtyřjádrovými procesory.

3.2. Softwarové požadavky

SharePoint pro svůj běh vyžaduje další software z produkce společnosti Microsoft jako je Windows Server nebo SQL Server. Přesný popis softwarových požadavků je opět převzat z oficiálních doporučení společnosti Microsoft, která je možné najít v online *Technet Library* [8].

Pro databázový server je možné využít *Microsoft SQL Server 2008 R2*, *Microsoft SQL Server 2008* s nainstalovaným *Service Pack 1* a *Cumulative Update 2* nebo *Microsoft SQL Server 2005* s nainstalovaným *Service Pack 3*. Ve všech případech se musí jednat o 64 bitovou verzi.

Pro *web front-end* servery a aplikační servery je nutné použít jakožto operační systém *Windows Server 2008* se *Service Pack 2* nebo *Windows Server 2008 R2* nebo *Windows Server 2008 R2* s nainstalovaným *Service Pack 1*. Vždy se musí jednat o 64 bitovou verzi a systém může být ve verzích *Standard*, *Enterprise*, *Data Center* nebo *Web Server*.

3.3. Možné architektury SharePoint farmy

3.3.1. Nasazení na jeden server

Nasazení všech potřebných komponent pro běh platformy SharePoint na jeden server nebývá v praxi příliš časté, používá se v podstatě pouze pro ukázku nebo vývoj řešení.

3.3.2. Nasazení na více serverů

Jakožto minimální konfigurace pro reálný běh systému SharePoint se uvažuje použití dvou serverů, přičemž jeden server slouží jako webový i aplikační a na druhý server je použit jakožto databázový. Toto řešení se doporučuje použít, pokud se systémem bude pracovat maximálně 10 tisíc uživatelů. [9].

Řešení je možné rozšířit o další server, který bude fungovat také jako aplikační i web server. Přidáním určitého *network load balancing* (dále jen NLB) řešení získáme poměrné odolný systém vůči výpadkům. NLB musí být nakonfigurován na persistentní sezení, protože *cache* není v sytému SharePoint sdílena napříč webovými servery [4].

Dále je možné řešení rozšířit o zvláštní aplikační server. Zůstanou tedy dva webové (*front-end*) servery, jeden aplikační server pro služby systému SharePoint a jeden databázový server.

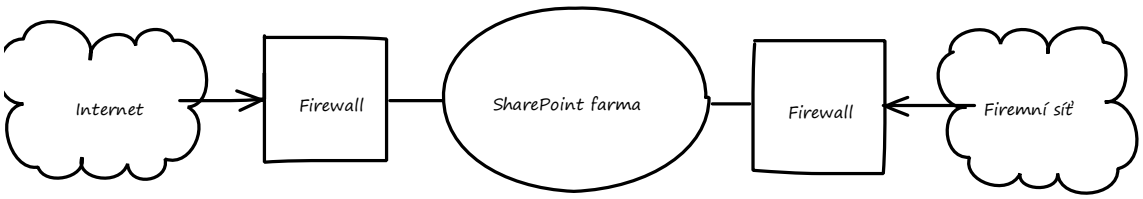
Při navyšování nároků na řešení je doporučeno přidávat jeden webový server na každých dalších 10 tisíc uživatelů. Doporučuje oddělení vyhledávací (query) a indexovací (crowl) služby na vlastní server (případně servery) od ostatních aplikačních služeb [5].

Při rozsáhlých řešeních je doporučeno rozdělit servery do několika skupin. V každé skupině poté běží jedna služba, případně více souvisejících služeb [5]. Takto může například vzniknout skupina serverů, na kterých běží indexovací (*crowl*) služba a skupina serverů pro vyhledávací (*query*) služby. Stejným způsobem je doporučeno rozdělit databázové servery. Může tedy například vzniknout skupina pro *Content* databáze a skupina pro *Search* databáze [5].

3.3.3. Extranetová řešení

Při extranetových řešeních se kromě optimálního počtu serverů a určení jejich rolí řeší také otázka bezpečnosti přístupu uživatele z vnějšku do interní sítě. V internetové knihovně *Technet* [10] společnosti Microsoft lze nalézt popis šesti topologií, z nichž ve zkratce přiblížím jedno často používané jednoduché řešení.

Řešení nazývané ***Back-to-back perimeter*** je založeno na rozdělení topologie do tří částí. SharePoint farma je umístěna v části (perimetru), která je oddělena pomocí firewallu od zbytku interní sítě a druhým firewallem od internetu. Požadavky jak interních tak externích pracovníků pro práci s aplikací prochází jedním ze dvou firewallů. Tímto je dosaženo určitého zabezpečení, protože extranetovým přístupem může být napadena pouze část interní sítě nacházející se v definovaném perimetru.



Obrázek 2 Velmi jednoduchý nákres architektury Back-to-back perimeter

4. Konfigurace a nasazení

4.1. Funkcionality (features)

SharePoint je koncipován jako rozšiřitelná platforma. Proto obsahuje koncept *features* neboli funkcionalit. Funkcionalita je definována souborem Feature.xml a jedním nebo více dalšími XML soubory. Tyto soubory popisují rozšiřující funkcionalitu, definují soubory, které mají být v rámci funkcionality nasazeny. Těmito soubory mohou být aspx stránky, šablony nebo třeba CSS styly a JavaScript [1]. Mezi funkcionalitami je možné definovat závislosti.

K funkcionalitě také může být připojen takzvaný *EventReciever*, což je programový kód, který reaguje na události vyvolávané při nasazování funkcionality. Těmito událostmi jsou *FeatureActivated, FeatureDeactivating, FeatureInstalled, FeatureUninstalling, FeatureUpgrading* a jsou vyvolávány během procesu nasazování funkcionality. Využívání těchto událostí pro provádění nějakým změn v systému SharePoint je velmi časté. Mohou být použity například pro zaregistrování nějakého

rozšiřujícího kódu jakým může být *Timer Job*, viz kapitola *5.6.1. Timer Job pro systém SharePoint a jeho vytvoření*.

Funkcionality jsou vytvářeny pro určitou logickou úroveň v systému SharePoint, kterými jsou *Farm*, *Web Application*, *Site Collection*, *Site*, viz kapitola *1.2. Logická struktura systému SharePoint.*

Ukázka definice funkcionality

Kód následující funkcionality pochází z vytvořeného intranetového řešení. Jedná se o velmi jednoduchou funkcionalitu, která rozšiřuje jedno ze standardních menu systému SharePoint o novou položku.

Obsah souboru MenuExtensions.feature:

```
<Feature xmlns="http://schemas.microsoft.com/SharePoint/" Title="Tieto Intranet
MenuExtensions" Id="545b3d37-07bd-4098-a531-7c2ecfeddc18" Scope="Site">
	<ElementManifests>
		<ElementManifest Location="CustomAction\Elements.xml" />
	</ElementManifests>
</Feature>
```

V této definici je vidět specifikace základní vlastností funkcionality jako je její název a jedinečné ID. Atribut Scope určuje pro jakou oblast bude funkcionalita aktivována. Trochu matoucí může být, že hodnota „Site“ určuje oblast s*ite collection*. Pro oblast web site slouží hodnota Web.

Dále obsahuje definice a odkazy na další soubory, v tomto případě pouze na jeden, Elements.xml. Tento soubor obsahuje již skutečnou definici položky pro rozšíření menu. Podoba tohoto souboru i s vysvětlením je uvedena v kapitole *5.7.1. Prvky uživatelského rozhraní*.

4.2. Řešení (Solutions)

Vlastní řešení vytvořené pro platformu SharePoint může být zabaleno do takzvaného balíčku řešení (*solution package*). Jedná se o soubory s příponou *wsp*. Balíčky obsahují DDF (data definition file) definici a jsou zkomprimovány pomocí CAB formátu [1].

V rámci jednoho řešení bývá obvykle zabaleno více funkcionalit, aplikační stránky a jiné soubory, všechny potřebné *assembly* atd.

4.3. Modely spouštění řešení (Executions models)

Systém SharePoint 2010 podporuje dva typy řešení, kterými jsou *sandbox* a *farm solution*.

4.3.1. *Farm* řešení

Tato řešení mají neomezený přístup k serverům ve farmě, mohou manipulovat se soubory ve file systému, spouštět jakýkoliv kód. Tento typ řešení je starší a byl představen již ve verzi 2007. Tato řešení jsou spouštěna ve stejném procesu jako samotná SharePoint aplikace (w3wp.exe procesy) [2].

4.3.2. *Sandbox* řešení

Jedná se o nový koncept pro vytváření řešení pro platformu SharePoint 2010. Tato řešení mají omezené možnosti. Nemohou například manipulovat se soubory ve *file* systému na serverech a využívat určité části SharePoint API. Výhodou je, pokud se některé z řešení chová nestandardně (například spotřebovává příliš mnoho zdrojů na serveru) je automaticky dočasně vypnuto. Administrátor farmy může také monitorovat tato řešení s použitím centrální administrace systému SharePoint. *Sandbox* řešení jsou spouštěna v rámci speciálního procesu *SPUCHostService.exe*, ne tedy *w3wp.exe* jako je tomu u *Farm* řešení [2].

Pokud je nutné využít nějakou část SharePoint API, která je pro Sandbox řešení nepřístupná, musí se použít princip *full trust proxies*. Myšlenka je taková, že architekt řešení navrhne určité bezpečné API, které mohou jednotlivá *Sandbox* řešení využívat. Toto API je dále implementováno v podobě takzvané *full trust proxy*, což je třída rozšiřující bázovou třídu SPProxyOperation. Takto vzniklá knihovna se silným jménem musí být nahrána do GAC a *proxy* třída zaregistrována do systému SharePoint (například pomocí příkazů *PowerShall*) [2].

Více o problematice Sandbox řešení, vytváření *full trust proxy* a přesných omezeních se lze dočíst například v knize *Microsoft SharePoint 2010, Building Solutions for SharePoint 2010* [6].

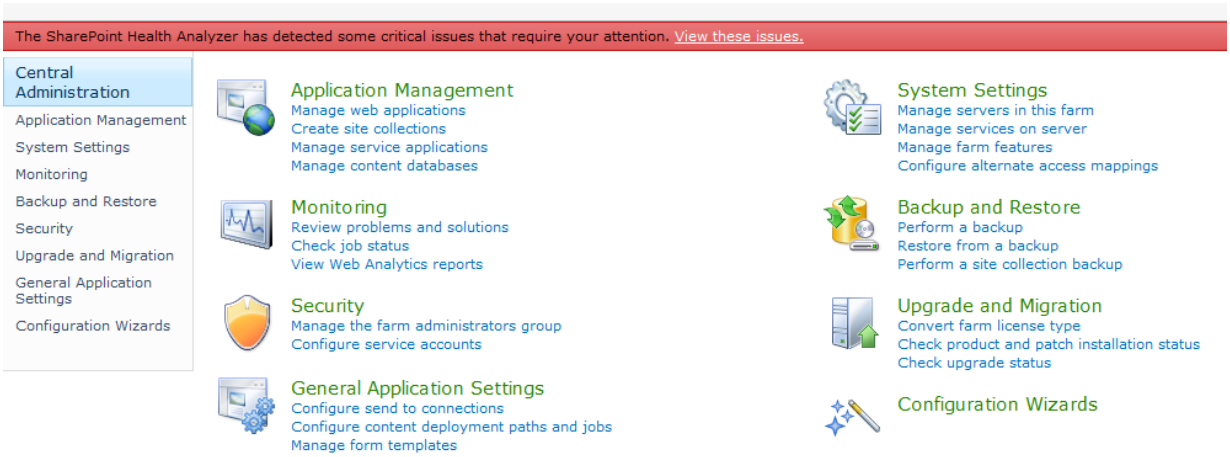
Intranetové řešení, které jsem implementoval, je vytvořeno jako ***Farm* řešení**, protože v tomto řešení jednak využívám části API, které není možné v *Sandbox* řešeních volat přímo a jednak kvůli vytvořeným aplikačním ASP.NET stránkám, které jsou nahrávány do *file* systému serveru.

4.4. Konfigurace systému SharePoint

Jelikož je SharePoint skutečně rozsáhlým řešením, je také jeho konfigurace velmi rozsáhlým tématem, o němž již byly napsány celé knihy. **V této práci popíši pouze několik nejzákladnějších znalostí z konfigurace sytému SharePoint**. Jedná se o znalosti, o kterých každý programátor vyvíjející řešení na platformě SharePoint musí mít alespoň základní povědomí. Administrátoři systému SharePoint musejí znám mnohem více.

4.4.1. Centrální administrace

Po instalaci systému je vytvořena jedna speciální webová aplikace, centrální administrace, sloužící ke konfiguraci systému. Podoba centrální administrace je vidět na obrázku níže (Obrázek 3).



Obrázek 3 Centrální administrace systému SharePoint

První sekce, *Application Management*, umožňuje vytvářet a konfigurovat webové aplikace a *site collections*. Při vytváření nové webové aplikace je možno mimo jiné nastavit aplikační pool, databázový sever a název databáze. Dále se zde nachází nastavení týkající se zabezpečení (povolení anonymního přístupu, metoda autentizace). Při vytváření nové *site collection* je zvolena šablona, podle které bude vytvořena root *web site* a dále také primární a sekundární administrátor *site collection.* V sekci *Application Management* se také nachází konfigurace aplikačních služeb jako je například *Search Service* mebo *Managed Metadata Service*. SharePoint 2010 umožňuje u těchto služeb nastavit jednotlivým službám různé administrátory, kteří takto získají přístup do centrální administrace k dané službě. Dále je možné konfigurovat, které služby budou dostupné nebo nedostupné pro určitou webovou aplikaci. V systému SharePoint 2010 je také možné tyto služby sdílet mezi více farmami. Může být také vytvořena SharePoint farma čistě za účelem poskytování těchto služeb jiným farmám. Tímto vznikají zajímavé možnosti z hlediska návrhu architektury rozsáhlých řešení a škálovatelnosti [2].

V sekci *System Settings* jsou dostupné funkce pro konfiguraci serverů ve farmě, jejich rolí a služeb, které na nich běží. V této sekci se také nachází nastavení *Alternate access mappings*, tedy nastavení mapování URL na jednotlivé webové aplikace.

Sekce *Security* se samozřejmě týká zabezpečení a přístupů. Mimo jiné je zde možnost spravovat administrátory farmy, účty pod kterými běží služby nebo nastavení antiviru.

Sekce *Monitoring* obsahuje různé nástroje pro monitorování řešení a případných problémů. Kromě toho také obsahuje seznam časových úloh (*Timer Jobs*) a umožňuje jejich okamžité spuštění. To je zapotřebí také při vývoji, kdy je potřeba otestovat nebo ladit nějakou vytvořenou úlohu.

Ze sekce *General Application Settings* bych vyzvedl především možnost zakázat používání programu SharePoint Designer pro určité *site collection*. SharePoint Designer je velmi silný nástroj, nicméně nezkušený uživatel může tímto nástrojem způsobit velké škody.

4.4.2. Další možnosti konfigurace

Systém může být spravován a konfigurován také z příkazové řádky. Pro programátory je důležité znát alespoň způsob přidání řešení do farmy a jeho nasazení do webové aplikace. K tomu slouží příkaz stdadm s patřičnými parametry.

Příkazem

stdadm –o addsolution –filename customsolution.wsp

se provede přidání řešení do farmy. Pro nasazení řešení je potřeba jako parametr *o* uvést *deploysolution*, specifikovat název řešení a webovou aplikaci:

stdadm –o deploysolution –name customsolution.wsp –allowGacDeployment –immediate –url http://SharePoint

Pro opakované administrační nebo konfigurační úlohy se obvykle používají skripty v jazyce *PowerShell*.

5. Implementace intranetového řešení

Budované intranetové řešení musí umožňovat vytváření publikačních *web site*, blogů, WIKI a týmových *web site*. Každý uživatel intranetu by měl mít možnost vytvořit si svůj vlastní blog a WIKI. Publikační a týmové site nebudou moci vytvářet všichni uživatelé, pouze uživatelé v určitých rolích. Všechny tyto site šablony již SharePoint obsahuje. Řešení samozřejmě chci obohatit funkcionalitou, kterou SharePoint neobsahuje a přinést tím zákazníkům určitou přidanou hodnotu. Bude se jednat především o tyto funkcionality:

1. Označování a ukládání oblíbených *web site*.
2. Agregování dat z oblíbených *web site* určitého typu, například z blogů a WIKI.
3. Vytváření blogů a WIKI dostupné všem uživatelům.
4. Umožnění uživatelům snadné procházení blogů a WIKI.
5. Implementace logiky pro načtení novinek ze všech publikačních site, ke kterým má uživatel přístup. Implementace logiky pro načtení úkolů ze všech týmových site, které se vztahují k uživateli.
6. Vytvoření prvků uživatelského rozhraní ve formě *web parts*, které umožní zobrazení agregovaných dat jednotlivých uživatelů, například na úvodní stránce intranetu.
7. Implementace logiky a uživatelského rozhraní pro načítání a zobrazení dat z veřejných zdrojů.
8. Implementace *workflow* se specifickou logikou pro revizi a publikování vytvořené stránky.

Protože se jedná o řešení společnosti Tieto bude také přizpůsoben grafický vzhled výsledné aplikace. Proto v rámci intranetového řešení implementuji také:

1. Vlastní šablonu pro publikační *web site*.
2. Upravení šablon pro blogy a WIKI.
3. Několik vlastních master page, které budou použity na výše zmíněných šablonách.
4. Několik šablon stránek.

Při návrhu architektury je nutné zohlednit, že řešení musí být snadno rozšířitelné a pozměnitelné. Vybudovaná aplikace bude sloužit spíše jako framework a základ pro konkrétní implementace zákaznických řešení než jako řešení, které by se beze změn pouze nasazovalo.

5.1. Architektura budované aplikace

Aplikaci jsem se rozhodl vybudovat na několikavrstvé doménově orientované architektuře [3]. Aplikace je rozdělena do tří hlavních vrstev: prezentační, doménové (logika aplikace) a vrstvy zajišťující persistenci dat. Objekty prezentační a doménové vrstvy komunikují také s inversion of control kontejnerem a objekty pro logování. Tuto funkcionalitu označuji jako cross-cutting vrstvu [3].

Doménovou vrstvu, která je jádrem celé aplikace, jsem dále rozdělil do několika modulů, konkrétně na:

1. Publikační modul
2. WIKI modul
3. Modul pro funkcionality blogů
4. Modul agregačních funkcionalit
5. Modul pro týmovou spolupráci

Doménová vrstva dále obsahuje definice rozhraní pro třídy datové vrstvy. **Aplikaci jsem navrhl tak, aby třídy implementované v rámci doménové vrstvy nezávisely na konkrétní implementaci vrstvy pro persistenci dat**. Závisí pouze na rozhraní, které může být implementováno nad libovolným úložitěm.

Stejně jako doménová vrstva není závislá na konkrétní implementaci datové vrstvy není ani prezentační vrstva závislá na konkrétních implementacích servisních tříd doménové vrstvy, závisí pouze na rozhraních, která mohou být implementována různě.

5.1.1. Využití Unity frameworku pro Dependency injection

Jak již bylo dříve zmíněno, aplikace musí být vytvořena tak, aby se dala snadno pozměnit a rozšířit. Například není žádoucí, aby nebylo možné implementovat jiný způsob ukládání dat a ten dále použít

v této aplikaci. Jako výchozí ukládání dat jsem zvolil ukládání pomocí systému SharePoint. Často je ale potřeba pro některá data použít externí databázi, například z důvodu zvýšení výkonu. Proto implementované třídy v aplikaci nejsou závislé na konkrétních implementacích jiných objektů, ale pouze na určitém definovaném rozhraní. Řešení jsem proto vytvořil s použitím vzoru dependency injection.

Pro získání konkrétní implementace k danému rozhraní jsem použil Inversion of Control framework Unity [11]. Framework Unity je oficiálně doporučován jako IoC framework společnosti Microsoft a vyvíjen pod hlavičkou *Microsoft Patterns and Practicies*, což byl důvod, proč jsem právě tento framework zvolil.

Díky tomu, že jsem při implementaci vytvářel závislosti pouze na určitém rozhraní, může být konkrétní implementace změněna pouhou změnou nastavení mapování mezi rozhraním a implementací. Například mohu nahradit implementaci repository třídy, která pro ukládání využívá systém SharePoint, za implementaci, která ukládá data do databáze. Nastavení mapování mezi rozhraním a implementací jsem umístil do souboru Web.config.

Nastavení *Unity* frameworku

Nastavení *Unity* frameworku jsem umístil do souboru *web.config* v sekci <unity>, což je standardní způsob konfigurace ve webových aplikacích. Při tomto výchozím umístění konfigurace je velmi snadné její načtení v *Unity* kontejneru, stačí zavolat metodu *LoadConfiguration* bez jakýchkoliv parametrů.

Ukázka části konfigurace:

```
<unity xmlns="http://schemas.microsoft.com/practices/2010/unity">
	<assembly name="MOSK.Intranet.Core, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=2455b6380b687b1a" />
	<container>
		<register type="Tieto.Intranet.Core.Domain.Blogs.IBlogsService"
mapTo="Tieto.Intranet.Core.Domain.Blogs.BlogsService">
			<constructor />
			<property name="SiteTemplateId">
				<value value="10001" />
			</property>
		</register>
	</container>
</unity>
```

V XML konfiguraci je vidět nejprve nastavení výchozího sestavení pomocí elementu *assembly*. Všechny typy budou tedy hledány v tomto sestavení, pokud u nich nebude uvedeno explicitně jiné. Elementem *register* je mapováno určité rozhraní na konkrétní implementaci, která má být pro toto rozhraní vytvořena. V tomto případě je rozhraní *IBlogsService* vrácena instance typu *BlogsService*. Pro tento typ je pomocí elementu *property* také nastavena výchozí hodnota vlastnosti *SiteTemplateId*.

5.1.2. Datová vrstva

Jako úložiště dat v této implementaci používám především seznamy systému SharePoint. Data ze seznamů SharePoint dále ukládá do MS SQL databáze. Pro přístup k datům systému SharePoint existují dva přístupy. Prvním je dotazování pomocí jazyku CAML. Dotazy v tomto jazyce jsou vytvářeny v kódu jakožto řetězec a vyhodnocovány přes API systému SharePoint. Od verze 2010 systému SharePoint je možné využít nový přístup, *Linq to SharePoint*, který přináší možnost objektového přístupu k datům.

Pro implementace této aplikace jsem zvolil nový přístup pomocí *Linq to SharePoint.* Pro jeho použití je zapotřebí vytvořit jednak třídu představující datový kontext vztahující se k určité web site a jednak třídy entit, na které jsou mapována data. Přesný postup použití *Linq To SharePoint* je popsán dále v kapitole *5.2.2. Vygenerování datových kontextů a entit*.

Pro přístup k datům jsem se rozhodl implementovat vzor repository. Rozhraní těchto tříd jsou umístěna v rámci doménové vrstvy, aby zůstala nezávislá konkrétní metodě persistence dat. Datová vrstva tedy obsahuje datové kontexty pro jednotlivé typy *web site* a implementace repository tříd, které budou umožňovat přístup k datům přes tyto datové kontexty. Třídy entit, na které jsou mapována data, jsou umístěny v rámci doménové vrsty protože se jedná o doménové objekty s určitou přidanou logikou.

Pro agregační funkcionality, týkající se WIKI a blogů, jsem implementoval také ukládaní do relační databáze. Díky důslednému oddělení vrstev stačilo pouze implementovat rozhraní příslušných repository tříd v nové *assembly* a provést úpravu mapování rozhraní na konkrétní implementaci v konfiguraci Unity frameworku. Přesnější popis implementace je uveden v kapitole *5.6. Agregace dat z oblíbených blogů a WIKI site*. Jakožto relační databázi jsem zvolil MS SQL Server 2008.

5.1.3. Doménová vrstva

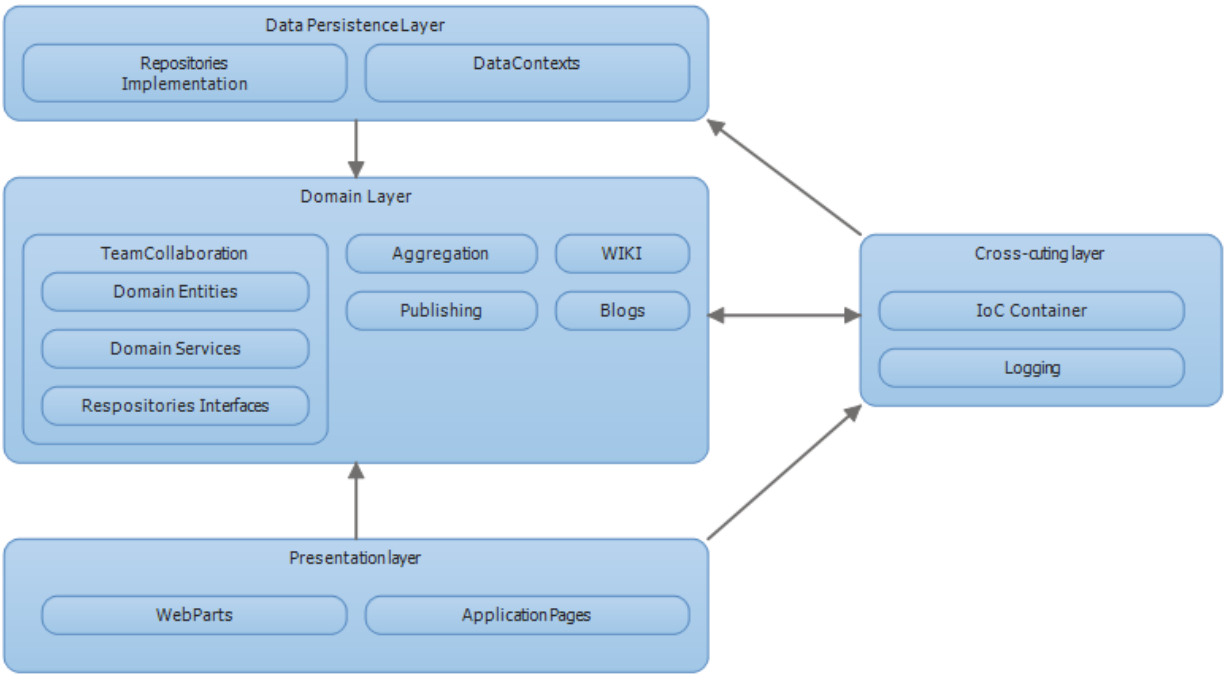
Jak již jsem zmínil, vrstvu doménové logiky jsem rozdělil na několik modulů, konkrétně na moduly týkající se publikačních funkcionalit, agregačních funkcionalit, blogů, WIKI a týmové spolupráce. Každý modul dále obsahuje třídy pro doménové entity, rozhraní doménových servisních tříd a výchozí implementace a rozhraní pro repository třídy. Doménové entity obsahují potřebnou doménovou logiku a jsou na ně mapována data. Doménové servisní třídy koordinují spolupráci mezi doménovými objekty a starají se o jejich načítání a ukládání pomocí repository tříd. Rozhraní pro repository třídy jsou implementována na datové vrstvě podle zvolené metody persistence dat.

5.1.4. Prezentační vrstva

Do prezentační vrstvy řadím implementace *web parts*, aplikačních stránek, master page šablon, šablon stránek (*page layouts*) a *site definition* šablony.

5.1.5. Diagram architektury aplikace

Následující diagram zobrazuje mnou zvolenou architekturu aplikace.



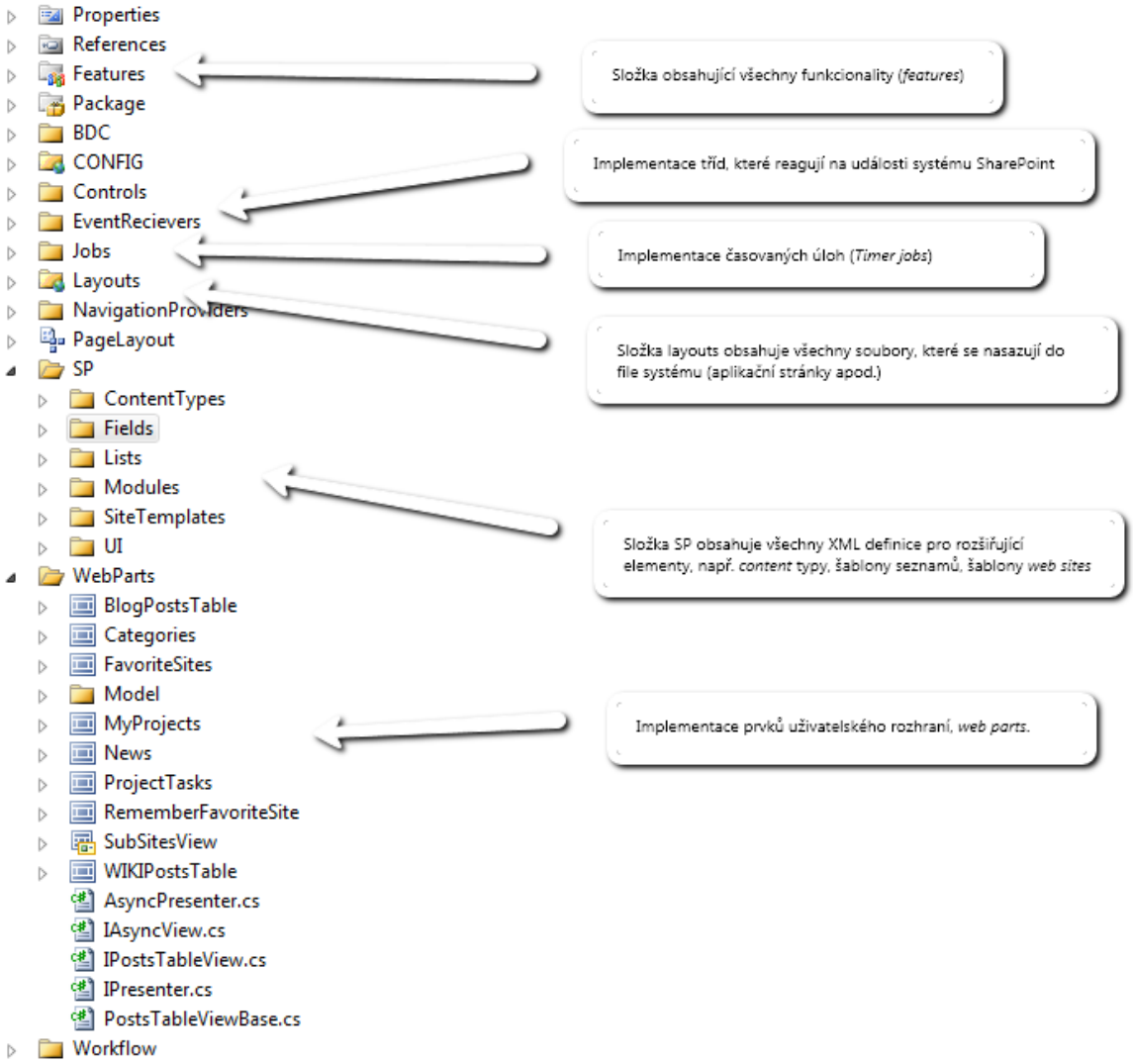
Obrázek 4 Architektura aplikace

Prezentační vrstva využívá vrstvu doménové logiky. Pro vytváření konkrétních doménových servisních objektů však používá Unity framework. Datová vrstva je závislá na doménové vrstvě, protože rozhraní repository tříd jsou obsažena v jednotlivých modulech doménové vrstvy. Každý z modulů doménové vrstvy obsahuje entitní třídy, doménové servisní třídy a rozhraní k repository třídám (pro jednoduchost vyznačeno v diagramu pouze u jednoho modulu).

5.1.6. Struktura projektu ve vývojovém prostředí MS Visual Studio 2010

Řešení jsem rozdělil do tří samostatných projektů. Prvním je projekt nazvaný Tieto.Intranet.14, který obsahuje prvky uživatelského rozhraní, všechny šablony pro *content* typy, seznamy, instance seznamů, site definice, sloupce a definice rozšiřující standardní prvky uživatelského rozhraní SharePoint. Název 14 je zvolen podle aktuální verze systému SharePoint.

Projekt dále obsahuje definice všech *feature* a definici výsledného WSP balíčku pro nasazení celého řešení.



Obrázek 5 Struktura projektu Tieto.Intranet.14

Druhý projekt nazvaný Tieto.Intranet.Core je pouze DLL knihovnou obsahující doménovou, datovou a *cross-cutting* vrstvu.

Třetí projekt, Tieto.Intranet.DBAccessLayer, je knihovna, která obsahuje implementace repository tříd určených pro agregaci příspěvků do relační databáze.

5.2. Vytvoření datové vrstvy v systému SharePoint

Jako úložiště dat jsem pro všechny funkcionality, které potřebují ukládat nějaká data, vytvořil seznamy v systému SharePoint, případně využil už exitující. Pro agregační funkcionality je navíc vytvořena i relační databáze.

5.2.1. Analýza požadavků vzhledem k datové vstrvě

5.2.1.1. Novinky a kategorizované stránky

Jedním z požadavků je možnost vytváření novinek a kategorizovaných stránek na publikačních *web site*. Samozřejmě potřebuji tyto data nějakým způsobem v systému SharePoint uložit a proto jsem potřeboval vytvořit:

1. Nový *content* type pro novinky, definici všech sloupců, které jsou v tomto content typu použity
2. Schéma seznamu pro novinky. Tento seznam bude podporovat ukládání položek výše zmíněného *content* typu.
3. Předpis pro vytvoření instance seznamu pro novinky
4. Nový *content* type představující kategorii. Tento typ musí obsahovat sloupec, kterým bude možné specifikovat odkaz na rodičovskou kategorii, aby mohly mít kategorie hiearchickou strukturu.
5. Pro uložení stránek jsem využil seznam Pages, který je v systému SharePoint již dostupný. Nicnémě do seznamu bylo zapotřebí přídat sloupec, který bude představovat odkaz na kategorii, do které stránka spadá.

5.2.1.2. Agregace příspěvků z blogů a WIKI

Uživatel má mít možnost označit si libovolnou *web site* jakožto oblíbenou, aby k ní přes implementované uživatelské rozhraní získal rychlý přístup. Jestliže se mezi těmito oblíbenými *web site* nachází blogy nebo WIKI budou se jejich nejnovější příspěvky agregovat do společného seznamu, aby mohly být uživateli rychle a přehledně zobrazeny.

Potřeboval jsem tedy nějakým způsobem uložit oblíbené stránky, agegované příspěvky z blogů a agregované příspěvky z WIKI. Proto jsem vytvořil:

1. *Content* typ představují oblíbené web site. K tomuto *content* typu jsem samozřejmě definoval několik sloupců, konkrétně pro uložení adresy *web site* a pro uložení ID jejího typu, aby bylo možné snadno poznat, zda se jedná například o WIKI. Pro uložení názvu používám sloupec Title, který je zděděn z bázového *content* typu Item. Dále používám sloupec pro zaznamenání uživatele, který si oblíbenou *web site* uložil.
2. Schéma seznamu pro oblíbené *web site* samozřejmě umožňuje ukládání položek výše zmíněného *content typu.*
3. Předpis pro vytvoření instance tohoto seznamu.

4. *Content* typ představující agregovaný příspěvek z blogu a *content* typ představují agregovaný příspěvek z WIKI. V obou případech musí být u položek těchto *content* typů možné uložit odkaz na položku uloženou v seznamu oblíbených *web site*, aby bylo jasné odkud příspěvek pochází.

5. Schémata seznamů pro agregované příspěvky z blogů a pro příspěvky z WIKI

6. Předpisy pro vytvoření těchto dvou seznamů

5.2.1.3. Datová vrstva pro blogy a WIKI

Pro tyto dva typy *web site* nepotřebuji vytvářet žádné další speciální seznamy. Využívám seznamů, které jsou definovány pro standardní blog a WIKI *web site*. Tyto standardní *web site* šablony jsem dále rozšířil, nicméně nepotřeboval jsem rozšiřovat datovou vrstvu.

S daty z blogů a WIKI ale samozřejmě ve svém řešení programově pracuji, proto jsem pro tyto typy *web site* také vytvořil datové kontexty, namapování seznamů na objekty a implementaci repository tříd (viz dále).

5.2.2. Umístění šablon content typů a seznamů v projektu Visual Studia 2010

Každý content type jsem umístil do samostatné složky. Obsahuje XML definici v souboru elements.xml. Stejně tak definice seznamů jsou tvořeny jednou složkou, která obsahuje definici i schéma seznamu. Ke každé definici seznamu je dále přiřazena složka „*NázevSeznamu*LI“ se souborem elements.xml, která obsahuje předpis pro vytvoření instance seznamu podle požadované šablony. Definice sloupců jsou umístěny ve společném souboru elements.xml ve složce FieldsDefinitions.

5.2.3. Vygenerování datových kontextů a entit

Datové kontexty a entity, na které jsou data mapována, jsem vytvořil pomocí nástroje spmetal.exe [12], který je součástí instalace systému SharePoint. Tento nástroj dokáže vygenerovat datový kontext a entity podle seznamů a *content* typů určité *web site*. Nástroj je schopný vygenerovat kód v jazycích C# nebo Visual Basic. Spmetal.exe je umístěn ve složce *bin* v adresáři, kde je nainstalován SharePoint. Obvykle se tedy jedná o adresář *%Program Files%\Common Files\Microsoft Shared\web server extensions\14\bin* [1]. Při použití spmetal musíme specifikovat především adresu SharePoint site, pro kterou chceme vygenerovat datový model a umístění na disku, kam se mají kódy uložit.

Příklad použití spmetal.exe z příkazové řádky:

spmetal /web:*http://adresa-SharePoint-webu* /code:*C:\temp\context.cs*

SPmetal podporuje i další parametry, kompletní soupis je možno najít na MSDN [12].

5.2.3.1. Vygenerované třídy

Nástroj SPMetal vygeneruje třídu datového kontextu, jejíž bázovou třídou je DataContext obsažená v knihovnách Linq to SharePoint. Při výchozím nastavení nástroje třída obsahuje veřejnou vlastnost pro každý neskrytý seznam na zvolené site typu EntityList<*TypEntity*> pro přístup k datům. Třídy datových kontextů jednotlivých typů *web site* jsem umístil do **datové vrstvy** budované aplikace.

SPMetal dále vygeneruje třídy entit, které umožňují objektově orientovaný přístup k datům uloženým v *content* databázi [1]. Tyto třídy jsou generovány jako *partial* [1], což umožňuje snadné doplnění dalších členů a logiky bez nutnosti zasahovat do vygenerovaného kódu. Tyto třídy jsem umístil do **doménové vrstvy** a dále jsem je rozšířil o požadovanou doménovou logiku.

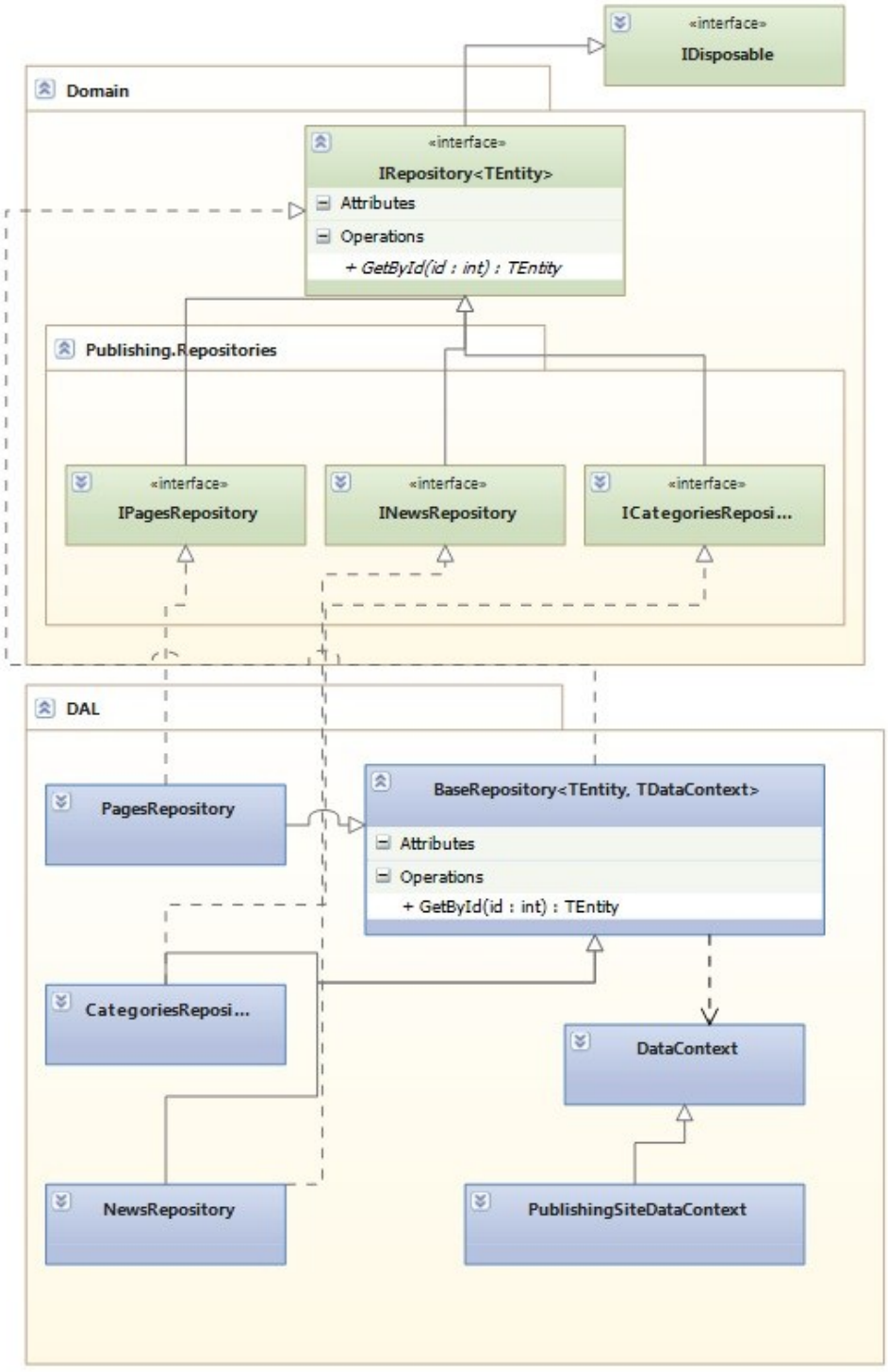
5.2.3.2. Vzor Repository

Pro přístup k datům jsem vytvořil třídy podle vzoru Repository. Vzor Repository slouží k zapouzdření logiky potřebné pro přístup k datům aplikace [3]. Každá z těchto tříd obsahuje referenci na potřebný DataContext vygenerový pomocí LINQ to SharePoint a poskytuje metody pro načítání, úpravu, ukládání a mazání dat. Jak jsem zmiňoval, rozhraní jednotlivých repository tříd jsou umístěna na doménové vrstvě a jejich implementace na datové vrstvě. Určité metody, které jsou společné pro všechny repository třídy, jsou umístěny v bázové třídě (například metoda GetItemById).

Objekt datového kontextu implementuje rozhraní IDisposiable a je doporučeno udržovat jeho instanci co nejkratší dobu [6]. Proto také v Repository třídách implementují rozhraní IDisposiable. V metodě *Dispose* je uveden kód sloužící k odstranění datového kontextu. Používání repository tříd na doménové vrstvě je pak vždy prováděno v rámci bloku using, aby došlo po ukončení práce s datovým kontextem co nejdříve k jeho zlikvidování pomocí metody Dispose. Ukázka kódu:

```
using (var repository = IoCUtility.ResolveRepository<IWikiPagesRepository>(siteUrl))
{
       // Práce s repository
}
```

Na následujícím obrázku (Obrázek 6) uvádím pro ukázku class diagram repository rozhraní a tříd používaných pro práci s daty na publikačních web site. Na tomto class diagramu je vidět jakým způsobem jsem navrhnul implementaci tříd, které zastřešují přístup k datům v systému SharePoint. Datové vrtvy pro další moduly aplikace (blog, WIKI, agregační) jsou tvořeny obdobnými třídami jako tomto případě publikačního modulu.



Obrázek 6 Class diagram repository rozhraní a tříd publikačních web site

5.2.4. Implementace *content* typů a seznamů

Jak jsem již popisoval v kapitole *5.2.1. Analýza požadavků vzhledem k datové vstrvě*, potřeboval jsem vytvořit celkem pět vlastních seznamů. Seznamy a *content* typy v systému SharePoint je možno vytvořit několika způsoby. Prvním je pomocí uživatelského rozhraní, což byl pro mou práci samozřejmě nepoužitelný způsob, protože řešení má být opakovaně nasaditelné bez nutnosti předem nějakým způsobem konfigurovat cílové prostředí. Dále je možné vytvoření programovým kódem anebo pomocí XML definic.

Ve své práci jsem se rozhodl použít XML definice, pouze relace mezi seznamem kategorí a stránek je vytvořena programově. Vytvoření úložiště dat ve formě seznamu se vždy skládá z následujících kroků:

1. Definování *content* typu položek, které budou ukládány. *Content* typy obsahují sloupce, které je potřeba také definovat.

   Vytvoření *content* typu před vytvořením seznamu není nutné, sloupce mohou být specifikovány přímo v rámci schématu seznamu. Vytváření *content* typů pro různé ukládané položky je silně doporučováno, například z důvodu znovupoužití *content* typu v jiném seznamu. Proto i já ve své práci pro všechny druhy ukládaných dat vytvářím vlastní *content* typ.

2. Vytvoření definice a schématu seznamu. Seznam je nutné propojit s dříve vytvořeným *content* typem.

3. Definování instance seznamu, která bude vytvořena na lokaci v systému SharePoint.

4. Nasazení všech výše popsaných složek pomocí jedné nebo více funkcionalit (*feature*) do systému SharePoint.

V následující kapitole **popisuji podrobně vytvoření seznamu pro novinky a vytvoření relace mezi seznamem kategorií a stránek** pro vytvoření představy, jakým způsobem se definuje datová vrstva v systému SharePoint. **Podrobný popis vytvoření dalších seznamů a *content* typů již v práci neuvádím.**

5.2.5. Datová vrstva pro publikační *web site*

Na publikačních webech, pro které jsem implementoval novou *site template* (bude popsána níže), jsou ukládány novinky a stránky řazené do určitých kategorií.

5.2.5.1. Novinky

Pro novinky jsem vytvořil nový content type News odvozený od content typu Item. Definice content typu:

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/SharePoint/">
  <!-- Parent ContentType: Item (0x01) -->
  <ContentType ID="0x01007b0cc8a75adb432fb80c1d86b2db1bd1"
               Name="News"
               Group="Tieto Intranet Content Types"
               Description="Content type for news"
               Inherits="TRUE"
               Version="0">
    <FieldRefs>

      <FieldRef ID="{24a82f87-dffb-4bba-8942-397ffd02e4e6}" Name="ShortDescription"
DisplayName="Short Description" Required="FALSE"/>
      <FieldRef ID="{206bfbcb-e439-4c57-9391-7cd9133b4710}" Name="Content"
DisplayName="Content" Required="TRUE"/>
      <FieldRef ID="{3bc82db5-26d9-4a78-b250-692801603f56}" Name="Created"
DisplayName="Created" Required="TRUE"/>

    </FieldRefs>
  </ContentType>

</Elements>
```

XML definice začíná tagem ContentType. V atributech elementu *ContentType* je specifikováno především jméno content typu, skupina, do které bude v rámci SharePointu zařazen, a jedinečný identifikátor content typu. Začátek tohoto identifikátoru určuje nadřízený content type. Nadřízeným content typem je základní typ Item obsažený v systému SharePoint, který má jedinečné id 0x01. Dále jsou v definici uvedeny reference na sloupce (v terminologii SharePoint se používá spíše název field), které content type obsahuje. Reference je dána jedinečným ID sloupce, dále je specifikováno jméno, které bude vidět v systému při zobrazení informací o content typu a popis sloupce. Nadřazený content type Item již obsahuje sloupec Title, takže jej nemusím přidávat pomocí reference.

Sloupce, které používám v content typu samozřejmě musí existovat. Pro specifikování sloupce se vytváří také XML definice. Definice pro sloupec Content:

```
<Field ID="{206bfbcb-e439-4c57-9391-7cd9133b4710}"
         Name="Content"
         DisplayName="Content"
         Type="Note"
         RichText="TRUE"
         RichTextMode="FullHtml"
         Group="Tieto Intranet Columns" />
```

V této definici specifikuji jedinečný GUID sloupce, název, název zobrazený v uživatelském prostředí SharePointu, datový typ sloupce a skupiny, do které bude sloupec zařazen. Jelikož se jedná o sloupec typu Note, tedy víceřádkový text, uvádím dále pomocí atributu RichText, že při vytváření hodnot pro tento sloupec se má zobrazit WYSIWYG editor. Atribut RichTextMode určuje, jaké funkce pro

formátování textu budou dostupné ve WYSIWYG editoru. Protože je hodnota nastavena na FullHtml budou uživatelé moci využít editor se všemi dostupnými funkcemi. Zbývající sloupce Created a ShortDescription jsou definovány obdobným způsobem. Sloupec Created je typu DateTime a sloupec ShortDescription typu Text.

Definovaný content type News je ale pouze předpisem pro určitý typ dat, ne samotným úložištěm. Proto pro novinky musíme dále vytvořit seznam, který bude podporovat ukládání dat typu News. Pro vytvoření definice nového seznamu jsou potřeba dva XML soubory.

1. Soubor elements.xml deklarující šablonu seznamu [2]

Ukázka definice seznamu:

```
<Elements xmlns="http://schemas.microsoft.com/SharePoint/">
    <!-- Do not change the value of the Name attribute below. If it does not match the
folder name of the List Definition project item, an error will occur when the project is
run. -->
    <ListTemplate
        Name="News"
        Type="10003"
        BaseType="0"
        OnQuickLaunch="TRUE"
        SecurityBits="11"
        DisplayName="News"
        Description="My List Definijgh tion"
        Image="/_layouts/images/itgen.png"/>
</Elements>
```

V definici je vidět ID šablony v atributu Type, které je nastaveno na 10003 a název šablony seznamu v atributu Name. Nastavení SecurityBits na hodnotu 11 znamená, že záznam v seznamu vytvořený určitým uživatelem může být tímto uživatelem také editován. Významy dalších atributů jsou jasné již z jejich názvů, nebudu je tedy více popisovat. Element ListTemplate podporuje více atributů než uvádím v této ukázce. Kompletní seznam je možné najít na MSDN [13].

2. Definice schématu seznamu je obsažena v souboru schema.xml. Schéma seznamu má následující kostru [1]:

```
<List>
  <Metadata>
    <ContentTypes />
    <Fields />
    <Views>
      <View>
        <Joins />
        <Projections />
        <ViewFields />
        <Query />
      </View>
    </Views>
    <Forms />
  </Metadata>
</List>
```

Jak je vidět z kostry, je možné specifikovat content typy, které mohou být ukládány v seznamu, pohledy na seznam (Views), které je dále možné využít při zobrazení dat v uživatelském rozhraní SharePointu nebo formuláře pro přidávání, editaci a zobrazení jednotlivých položek [1].

Při vytváření definice seznamu jsem použil šablonu dostupnou ve Visual Studiu, která automaticky vygeneruje defaultní nastavení pro Views a Forms. Tato nastavení jsem neměnil, ve schématu jsem pouze specifikoval vlastnosti seznamu v elementu List a přiřadil příslušný content type v sekci ContentTypes:

Ukázka upravené části definice schématu seznamu:

```
<List xmlns:ows="Microsoft SharePoint" Title="News" FolderCreation="FALSE"
Direction="$Resources:Direction;" Url="Lists/News" BaseType="0"
xmlns="http://schemas.microsoft.com/SharePoint/">
  <MetaData>
    <ContentTypes>
      <ContentTypeRef ID="0x01007b0cc8a75adb432fb80c1d86b2db1bd1">
      </ContentTypeRef>
    </ContentTypes>
```

V sekci ContentTypes pomocí elementu ContentTypeRef definuji asociaci mezi seznamem a dříve definovaným content typem News. Požadovaný content type je rozeznán podle atributu ID.

Předešlé XML soubory zajišťují pouze definici šablony a schématu seznamu, nevytvářejí samotnou instanci seznamu. Pro tento účel jsem vytvořil další soubor elements.xml umístěný ve vlastní složce NewsLI (LI = list instance). Soubor obsahuje element ListInstance, který určuje jaký seznam se má vytvořit a podle které šablony:

```
<Elements xmlns="http://schemas.microsoft.com/SharePoint/">
  <ListInstance Title="News"
                OnQuickLaunch="FALSE"
                TemplateType="10003"
                Url="Lists/News"
                Description="News">
  </ListInstance>
</Elements>
```

5.2.5.2. Kategorie

Pro kategorie jsem vytvořil také nový content type, nazvaný Category, obdobným způsobem jako v případě novinek, proto nebudu kódy definice znovu uvádět. Tento content type bude referencovat celkem tři sloupce: Title, Description a Parent. Sloupec Title je opět zděděn z bázového content typu Item a sloupec Description je typu Text. Sloupec Parent slouží k uložení asociace mezi rodičovskou a podřízenou kategorií, bude proto typu **Lookup**. Pomocí typu Lookup je možné v seznamu vytvořit referenci na záznam z jiného nebo stejného seznamu.

Ukázka definice:

```
<Field ID="{cbcb1dd4-8b03-4a33-9db6-a8317ed65138}"
       Name="Name"
       DisplayName="Parent Category: Name"
       Type="Lookup"
       Required="True"
       List="Self"
       ShowField="Title"
       Group="Tieto Intranet Columns"/>
```

Do atributu List se uvádí URL nebo případně ID seznamu, jehož položky chceme sloupcem referencovat. V případě, že se jedná o stejný list, v jakém je sloupec obsažen, uvede se slovo „Self“.

Při vytváření sloupce typu Lookup je velmi důležité, aby byl sloupec nasazován až v době, kdy seznam, na který sloupec odkazuje, existuje v systému SharePoint. Zdá se to být sice přirozené, ale v případě, že jsou sloupce i instance seznamů, na které odkazují, nasazovány v rámci jednoho řešení, je důležité pořadí aktivace features obsahující Lookup sloupce a instance listů. Feature obsahující definici Lookup sloupce musí být aktivována až po aktivaci feature, která vytváří požadovanou instanci seznamu. Další možností je nasazovat Lookup sloupec a instanci seznamu v rámci jedné feature. Vytváření Lookup sloupců je obecně poměrně problémová část při definování datové vrstvy, osobně jsem korektní vytvoření relací mezi seznamy řešil poměrně dlouhou dobu. Jak jsem zjistil, bývá často výhodnější tyto sloupce vytvářet programovým kódem, u kterého se snadněji zajistí, aby byl spustěn ve správnou dobu.

5.2.5.3. Stránky

Pro uložení stránek jsem použil již hotové publikační řešení platformy SharePoint, kterým je SharePoint Server Publishing Infrastructure. Tato funkcionalita je dostupná pouze v placené verzi systému

SharePoint. Pro použití této funkcionality musí být aktivována SharePoint Server Publishing Infrastructure na úrovni *site collection* a SharePoint Server Publishing na úrovni *web site*. Tyto feature budou automaticky aktivovány při vytvoření publikační site, přesný popis viz Site template pro publikační weby intranetu.

Po aktivaci této funkcionality je na *web site* vytvořeno několik seznamů, nejdůležitějším je seznam Pages sloužící k ukládání stránek. Do tohoto seznamu je potřeba přidat Lookup sloupec, který zajistí propojení mezi stránkou a kategorií. Pro přidání toho sloupce jsem vytvořil speciální funkcionalitu PublishingWebLists. Tato funkcionalita se aktivuje při vytvoření publikační site, ale až po aktivování funkcionality SharePoint Server Publishing. K této funkcionalitě jsem vytvořil *EventReciever*, ve kterém reaguji na událost FeatureActivated a programově přidávám do seznamu Pages sloupec Category.

Ukázka kódu pro přidání sloupce:

```
using (SPWeb web = (SPWeb)properties.Feature.Parent)
{
	if (!web.Lists["Pages"].Fields.ContainsField("Category"))
	{
		string fieldName = web.Lists["Pages"].Fields.AddLookup("Category",
web.Lists["Categories"].ID, web.ID, true);

		SPFieldLookup lookup = web.Lists["Pages"].Fields[fieldName] as
	SPFieldLookup;

		lookup.LookupField = "Title";
		lookup.Update();
		web.Lists["Pages"].Update();
	}

}
```

Při řešení potřebuji instanci objektu představující site, na kterou je feature nasazována. Tu získám z vlastnosti Parent, protože je feature nasazována na úroveň site. Dále zkontroluji, zda list Pages již sloupec Category neobsahuje. Kontrola se provádí z důvodu možné opětovné aktivace feature. Pokud sloupec ještě není v seznamu obsažen, vytvořím jej pomocí metody AddLookup. Jako jeden z argumentů se předává ID seznamu Categories, s nímž se vytváří asociace. U sloupce PCategory dále nastavuji, že má v uživatelském rozhraní zobrazovat hodnoty ze sloupce Title asociovaného seznamu. Nakonec je potřeba změny provedené nad sloupcem PCategory i seznamem Pages uložit pomocí metody Update.

5.2.5.4. Indexy v seznamech

Podobně jako v klasických databázových tabulkách je i v systému SharePoint možné nastavit určitý sloupec v seznamu jako indexovaný. Takovýto index ale není spravovaný databázovým serverem ale přímo systémem SharePoint. Indexace může významně navýšit výkon mnoha dotazů, jako například řazení a fitrování [6].

Pro nastavení indexování sloupce slouží atribut ***Indexed*** elementu ***Field***.

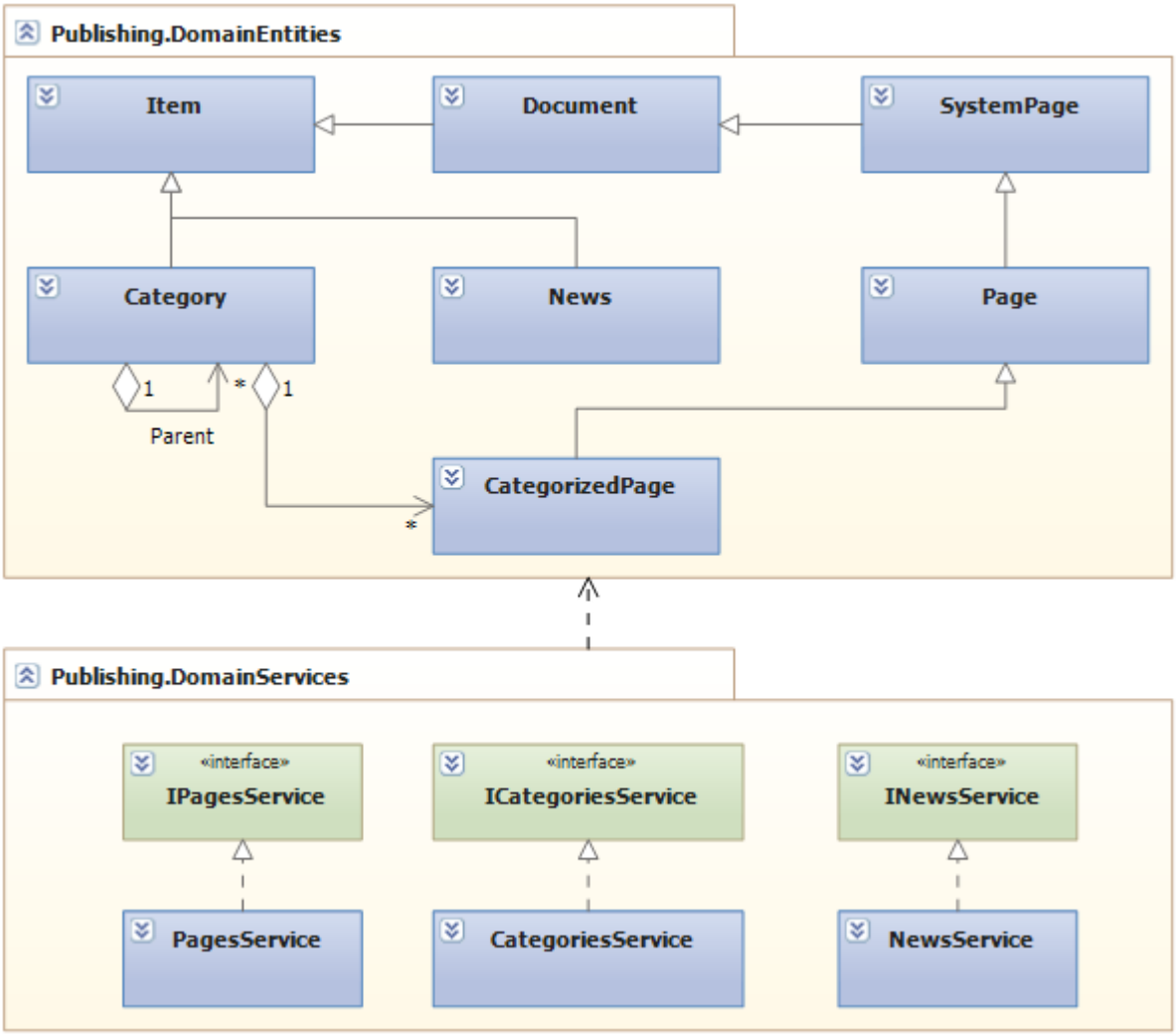
5.2.6. Mapování vytvořené datové vrstvy na objekty

V této kapitole uvádím class diagramy modulů doménové vrstvy, u kterých jsem využil možnost mapování datové vrstvy systému SharePoint na entity v jazyku C# pomocí technologie *LINQ to SharePoint*. V diagramech jsou také uvedeny servisní doménové třídy. Ty využivájí repository tříd pro ukládání a načítání entit a koordinují práci mezi nimi. **Rozhraní tříd repository v těchto diagramech již neuvádím.**

5.2.6.1. Publikační modul

Následující class diagram zobrazuje třídy modulu doménové vrstvy, který rozšiřuje a pracuje s publikační funkcionalitou systému SharePoint. Datová vrstva je mapována na třídy v balíčku DomainEntities.

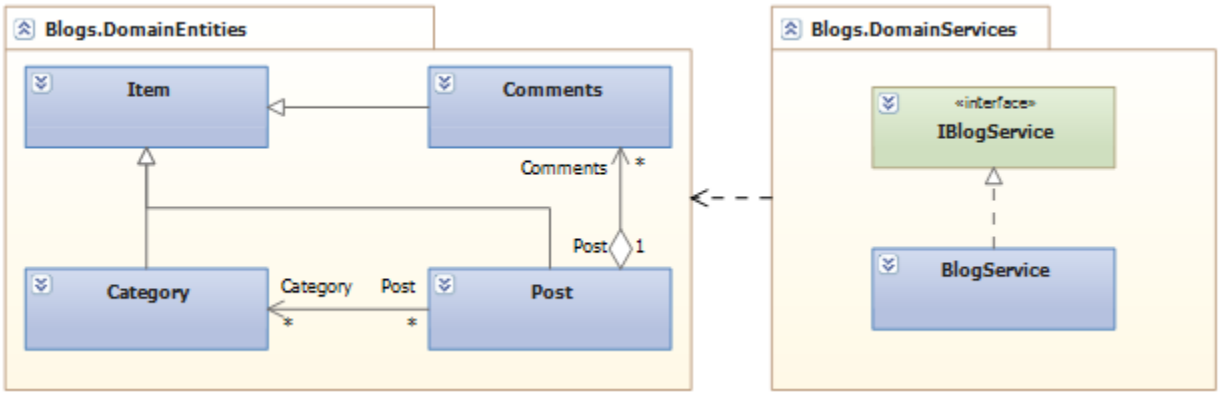
Třídy v balíčku DomainEntities odpovídají *content* typům, které jsou použity na publikační site v některém ze seznamů. Jak je vidět z class diagramu, všechny entity dědí z bázového typu Item. Content typy představující dokumenty dědí z bázového content typu Document. Content type Page jsem dále rozšířil na typ CategorizedPage.



Obrázek 7 Class diagram entit publikačních site

5.2.6.2. Modul pro funkcionality blogů

Pro web site typu blog jsem rozšířil již hotovou šablonu platformy SharePoint. Seznamy a content typy pro ukládání dat blogů jsou již v SharePointu hotové. Pro další práci s těmito daty ale bylo potřeba provést mapování na entity v jazyce C# stejně jako v předchozím případě. Následující diagram zobrazuje třídy v modulu *Blogs*, datová vrstva je opět mapována na třídy v balíčku DomainEntities.



Obrázek 8 Class diagram entit blog web site

5.2.6.3. WIKI modul

Stejně jako v případě blogů jsem pro WIKI rozšířil již hotové řešení systému SharePoint. Pro WIKI *web site* již SharePoint tedy také obsahuje datovou vrstvu. Tato datová vrstva je navíc velmi jednoduchá, obsahuje pouze jeden seznam pro ukládání WIKI stránek, neuvádím tedy ani class diagram.

5.2.6.4. Modul agregačních funkcionalit

Uživatel má mít možnost uložit oblíbené *web site*. Pokud jsou typu blog nebo WIKI, budou nejnovější příspěvky na těchto *web site* agregovány do speciálních seznamů, aby mohly být uživateli jednoduše zobrazeny. Pokud by příspěvky nebyly agregovány, mohlo by procházení různých *web site* a načítání příspěvků trvat velmi dlouhou dobu.

Pro tyto agregační funkcionality jsem vytvořit několik content typů a seznamů. Seznamy budou pro agregační funkce umístěny na root site intranetu.

Oblíbené web site

Pro ukládání oblíbených web site jsem vytvořit content type a definice sloupců stejným způsobem, jaký byl popsán v kapitole 5.2.3 a nazval jsem je FavoriteSites. Content typ FavoriteSites umožňuje uložit název *web site*, její adresu, typ a uživatele, ke kterému se oblíbená *web site* vztahuje. Sloupec pro uložení uživatele je speciálního datového typu **User**, který umožňuje uložení asociace mezi záznamem v seznamu a uživatelem v systému SharePoint. Typ uložené *web site* ukládám z důvodu, abych mohl jednoduše poznat, zda se jedná o *web site*, ze které se mají agregovat nejnovější příspěvky. Název *web site* ukládám do sloupce Title, který je zděděn ze základního typu Item. K tomuto content typu jsem dále samozřejmě vytvořil definici pro seznam a instanci seznamu.

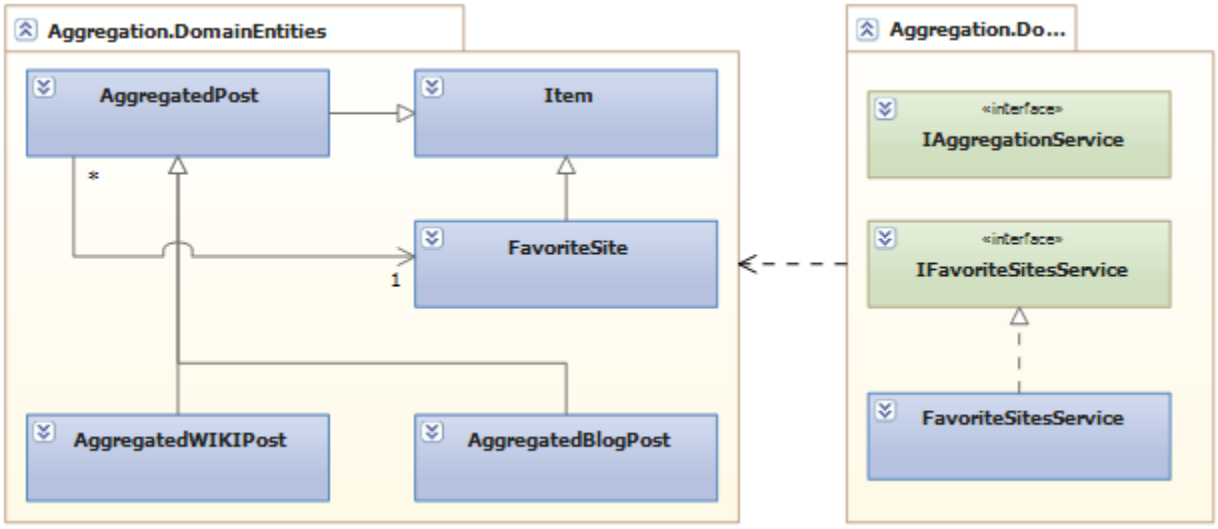
Mapování sloupců typu User

Nástroj spmetal vytvoří pro sloupec typu User ve výsledné třídě dvě vlastnosti. Jednu typu *string* a jednu typu *integer*. Při načtení dat obsahují vlastnosti typu *string* uživatelské jméno uživatele, zatímco *integer* vlastnost obsahuje jeho ID. Považuji za dobré zmínit, že při ukládání je potřeba vždy nastavovat ID uživatele, protože mne osobně toto překvapilo a zdrželo při implementaci. Nastavení uživatelského jména ukládání neovlivní. Pokud není nastaveno ID, relace mezi záznamem a uživatelem systému se neuloží.

Agregované příspěvky z blogů a WIKI

Pro agregaci příspěvků z blogů a WIKI jsem vytvořil jeden společný content type AggregatedPost, který umožní uložit název agregovaného příspěvku, jeho ID a *web site*, ze které byl získán. Pro uložení názvu příspěvku využívám opět sloupec Title, pro ID příspěvku jsem vytvořil speciální sloupec typu Number a sloupec pro uložení *web site* je typu Lookup a odkazuje na záznamy v seznamu FavoriteSites, ze kterého se dá vyčíst typ a adresa *web site*. Content type AggregatedPost je použit ve dvou seznamech, v seznamu AggregatedBlogPosts a AggregatedWikiPosts.

Stejně jako ve všech předchozích případech jsem datovou vrstvu namapoval na třídy jazyka C# pomocí LINQ to SharePoint. K rozhraní IAggregationService v balíčku DomainServices není vytvořena výchozí implementace, toto rozhraní je implementováno v jiných modulech. Konkrétně je implementováno třídou BlogsService a WikiService z modulu funkcionalit pro blogy respektive WIKI. Až tyto třídy zajišťují skutečnou agregaci požadovaných dat. Toto rozhraní má smysl pro další rozšiřování agregačních funkcionalit, viz dále kapitola *5.6. Agregace dat z oblíbených blogů a WIKI site*.



Obrázek 9 Class diagram entit pro agregační funkcionality

5.2.7. Jedinečné klíč složený z více sloupců v seznamu

Při ukládání oblíbených *web site* je jistě vhodné, aby nebyly uloženy duplicitní záznamy. Uživatelské rozhraní, které jsem vytvořil, samozřejmě nic takového nepovolí, nicméně je jistě dobrou praxí zavést toto omezení i na datové vrstvě. Jedinečným klíčem záznamu je v tomto případě identifikátor uživatele a identifikátor *web site.* Při použití klasické SQL databáze by byl tento problém snadno řešitelný pomocí složeného unikátního klíče, který by zahrnoval tyto dva sloupce. V systému SharePoint ale bohužel takováto možnost pro datovou vrstvu není. Nejlepším způsobem jak vyřešit tento problém, je využití událostí, které jsou vyvolávány při ukládání nové položky do seznamu. Je potřeba vytvořit *Event Reciever*, což je třída, která dědí ze *SPItemEventReceiver* třídy. V této třídě je dále potřeba přepsat metodu *ItemAdding*, která je volána před vložením položky do seznamu. Parametrem metody je objekt typu *SPItemEventProperties*, z nějž je možné získat hodnoty záznamu, který má být uložen a provést tak kontrolu, zda je záznam jedinečný. Má-li být vkládání zrušeno, stačí nastavit vlastnost *Cancel* tohoto objektu na *true*.

Event Reciever je nasazen v rámci určité funkcionality a jak je obvyklé pro různá rozšíření v systému SharePoint, je k tomu potřeba určitá XML definice.

Ukázka:

```
<Receivers ListTemplateId="10001">
     <Receiver>
       <Name>AggregatedBlogPostsEventRecieverItemAdding</Name>
       <Type>ItemAdding</Type>
       <Assembly>$SharePoint.Project.AssemblyFullName$</Assembly>
       <Class>Tieto.Intranet._14.EventRecievers.AggregatedBlogPostsEventReciever.Aggregat
edBlogPostsEventReciever</Class>
        <SequenceNumber>10000</SequenceNumber>
      </Receiver>
</Receivers>
```

V definici je vidět specifikace, jakého typu seznamů se tento *Event Reciever* týká (atribut ListTemplateId), dále pak jeho název, pro jakou událost má být volán (hodnota elementu Type), název třídy a sestavení, kterého je součástí.

5.3. Vytvoření vlastních site definition

Pro účely intranetového řešení jsem připravil čtyři *site definition*. Jedná se o definici pro publikační *web site*, pro blogy a pro WIKI. Čtvrtá šablona je určena pro web site, která bude sloužit jako rodičovská web site pro všechny blogy a WIKI. Šablony jsou vytvořeny tak, aby při vytvoření *web site* byly korektně aktivovány všechny funkcionality (*features*), které jsou pro běh *web site* nezbytné (funkcionality pro vytvoření odpovídajících content typů, seznamů atd.). Šablony také obsahují předpis pro vytvoření úvodní stránky default.aspx, aby bylo zajištěno automatické výchozí rozmístění některých web parts. U všech těchto *web site* šablon budou použity vlastní master page.

5.3.1. Site definition pro publikační web site

Pro každou *site definition* je potřeba vytvořit tři XML soubory (podle Site Definition Structure v SharePoint 2010 as Development platform [1]):

- webTemp.xml – obsahuje informace o šabloně, které se zobrazí v dialogu pro vybrání site template při vytváření nové web site.
- onet.xml – obsahuje jednu nebo více konfigurací pro *site definition*, které obsahují odkazy na *features*, které mají být aktivovány atd.
- default.aspx – stránka, která bude použita jako úvodní stránka vytvořené *web site*

5.3.1.1. webTemp.xml

Do souboru webTemp.xml jsou tedy umístěny základní informace o šabloně jako je název, popis atd.

Ukázka souboru webTemp.xml pro publikační web site:

```
<?xml version="1.0" encoding="utf-8"?>
<Templates xmlns:ows="Microsoft SharePoint">
  <Template Name="TietoPublishingSite" ID="10002">
    <Configuration ID="0" Title="Tieto Publishing Site"
                   Hidden="FALSE"
                   ImageUrl="/_layouts/images/CPVW.gif"
                   Description="Tieto Publishing Site"
                   DisplayCategory="SharePoint Customizations">
    </Configuration>
  </Template>
</Templates>
```

Z elementu Template lze vyčíst název šablony a její ID. Atributy elementu Configuration dále specifikují především titulek, popis a umístění ikony vztahující se k šabloně. Tyto informace jsou použity v systému SharePoint v dialogu pro vytvoření nové *web site*. Atribut ID elementu Configuration odkazuje na konfiguraci v souboru onet.xml [1].

5.3.1.2. onet.xml

Soubor onet.xml je jádrem šablony a specifikuje všechny komponenty, která má web site obsahovat jako seznamy, web parts, stránky. [1]

Ukázka části definice ze souboru onet.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<Project Title="Tieto Publishing Site" Revision="2" ListDir="" xmlns:ows="Microsoft
SharePoint" xmlns="http://schemas.microsoft.com/SharePoint/">
  <NavBars>
  </NavBars>
  <Configurations>
    <Configuration ID="0" Name="Tieto Publishing Site"
CustomMasterUrl="_catalogs/masterpage/publish.master">
      <Lists/>
      <SiteFeatures>

        <!-- Aggregation lists + instances -->
        <Feature ID="465138bd-5cbb-477c-9148-0f0846f6c323" />

        <!-- Publishing Site Infrastructure -->
        <Feature ID="f6924d36-2fa8-4f0b-b16d-06b7250180fa" />

        <!-- Custom Content Types-->
        <Feature ID="d86c7005-d31d-43ae-b86b-38f28b81072e" />

        <!-- Custom styles + images -->
        <Feature ID="cfac09fe-5aff-4b66-8c3c-22ec02af3af5" />

      </SiteFeatures>
```

V Configurations sekci jsou uvedeny jednotlivé možné konfigurace, kde každá může obsahovat odkazy na definice seznamů, features a modulů [1]. V sekci Lists neuvádím žádné definice seznamů, místo toho odkazuji na features, které obsahují definice potřebných content typů, seznamů a jejich instancí. V sekci SiteFeatures jsou uvedeny funkcionality, které mají být při vytváření web site aktivovány na oblast *site collection*. Features, které se mají aktivovat, jsou specifikovány pomocí jejich jedinečného ID. Při vytváření publikační *web site* je potřeba aktivovat Publishing Site Infrastructure funkcionalitu. Jak jsem již zmiňoval, pro vytváření publikačních stránek zčásti využívám funkcionalitu obsaženou v systému SharePoint, kterou dále rozšiřuji. Protože je tato *site definition* zamýšlena také pro výchozí *web site* v kolekci intranetu, aktivuji dále funkcionality, které obsahují definice content typů, seznamů a instancí seznamů pro agregaci dat. Tyto funkcionality budou aktivovány při vyváření první publikační *web site*, což by měla být domovská *web site* intranetu. Dále aktivuji funkcionalitu, která se postará o nasazení CSS stylů a obrázků pro vytvořenou master page. Protože jsou tyto styly a obrázky z velké části společné pro všechny master page, které jsem v rámci intranetového řešení vytvořil, nasazuji tuto feature na oblast *site collection*.

Stejným způsobem, jakým jsou aktivovány funkcionality pro oblast site collection, jsou aktivovány také funkcionality pro oblast *web site*. Umístěny jsou v sekci WebFeatures.

```
<WebFeatures>
        <!-- Master Page -->
        <Feature ID="4dae3485-c2e5-4c07-adcc-28a7417b6dd9" />
        <!-- New Page Action (in menu) -->
        <Feature ID="a54c4c42-5c0d-4ba8-b37d-1e288e337102" />
        <!-- Publishing -->
        <Feature ID="22A9EF51-737B-4ff2-9346-694633FE4416">
          <Properties xmlns="http://schemas.microsoft.com/SharePoint/">
Value="$Resources:osrvcore,List_Pages_UrlName;/default.aspx"/>
            <Property Key="SimplePublishing" Value="true" />
          </Properties>
        </Feature>

        <!-- Publishing Feature -->
        <Feature ID="94c94ca6-b32f-4da9-a9e3-1f3d343d7ecb" />

        <!--Lists definitions + instances-->
        <Feature ID="be3732cc-6f55-41c1-b62c-a24a43540fad" />

</WebFeatures>
```

Pro oblast *web site* aktivuji Publishing a PublishingWeb feature, které souvisí s publikační funkcionalitou systému SharePoint. Dále se aktivuje funkcionalita obsahující master page šablonu pro publikační *web site*, funkcionalitu obsahující definice seznamů a jejich instancí (kategorie, novinky) a také funkcionalitu, která rozšiřuje standardní uživatelské menu v systému SharePoint. (Toto rozšíření bude popsáno později).

Dále v Configuration specifikuji modul pro domovskou stránku vytvářené web site.

```
<Modules>
   <Module Name="Home" />
</Modules>
```

V Configuration sekci se nachází pouze reference na Home modul, samotná definice modulu má následující podobu:

```
<Module Name="Home" Url="$Resources:osrvcore,List_Pages_UrlName;" Path="">
      <File Url="default.aspx" Type="GhostableInLibrary" Level="Draft" >
        <Property Name="Title" Value="Home" />
        <Property Name="PublishingPageLayout"
Value="~SiteCollection/_catalogs/masterpage/defaultTemplate.aspx,
$Resources:cmscore,PageLayout_WelcomeLinks_Title;" />
        <Property Name="ContentType"
Value="$Resources:cmscore,contenttype_welcomepage_name;" />
```

V elementu Module je samozřejmě specifikováno jméno modulu. Pomocí modulu se nasazují soubory (v tomto případě pouze jeden – default.aspx), proto je v atributu URL specifikována adresa, kam se má soubor umístit. Protože je na web site použita Publishing funkcionalita, umísťuji tento soubor do seznamu Pages. Adresa seznamu je získána ze zdrojů systému SharePoint.

Pokud je dané *web site* použita Publishing funkcionalita je vždy nutné stránku default.aspx umisťovat do seznamu Pages. Publishing funkcionalita totiž nastaví výchozí stránku *web site* právě na soubor default.aspx v seznamu Pages. Další možností by bylo toto nastavení výchozí stránky po aktivaci Publishing funkcionality opět změnit pomocí určitého programového kódu, já ale považuji umisťování default.aspx stránky do seznamu Pages za dobrou praktiku.

Nejdůležitějšími atributy elementu File jsou URL, který specifikuje pod jakou adresou bude soubor přístupný, a Type. Hodnota GhostableInLibrary znamená, že soubor bude součástí seznamu, jehož bázovým typem je DocumentLibrary [14]. Dále jsou pomocí elementů Property nastavena některá metadata souboru. Konkrétně nastavuji titulek, šablonu stránky a *content type*, pod kterým bude dokument uložen do seznamu.

Protože nechci, aby po vytvoření web site byla úvodní stránka zcela prázdná, definuji pomocí elementů AllUsersWebPart komponenty (*web parts*), které mají být na stránku umístěny automaticky:

```
      <AllUsersWebPart ID="BlogPostsTable" WebPartZoneID="RightPanelWebPartZone"
WebPartOrder="1">
```

V elementu AllUsersWebPart je specifikováno ID komponenty (*web part*) a dále zóna, do které má být umístěna (atribut WebPartZoneID). Dále je určen samotný typ třídy pro *web part* a případně přednastaveny některé jeho vlastnosti, viz následující ukázka kódu:

```
<![CDATA[<webParts>
<webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
<metaData><type
name="MOSK.Intranet._14.WebParts.BlogPostsTable.BlogPostsTableView,MOSK.Intranet.14,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=d52e620de5621863" />
<importErrorMessage>$Resources:cmscore,WebPartImportError;</importErrorMessage>
</metaData>
   <data><properties>
                  <property name="Title" type="string">Newest Blog Post</property>
                  <property name="Description" type="string"></property>
                </properties></data>
             </webPart></webParts> ]]></AllUsersWebPart></File>
</Module>
```

Z předešlé ukázky XML definice je vidět, že třída, která představuje *web part*, musí být definována včetně kompletního názvu sestavení, který zahrnuje i verzi a veřejný klíč. Ve výpisu je také možné si všimnout nastavení dvou základních vlastností, *Title* a *Description*, které je možné nastavit u každé *web part* komponenty.

Na úvodní stránku umísťuji automaticky web parts pro:

- zobrazení novinek z *web sites*, u kterých má uživatel práva na prohlížení

- zobrazení nejnovějších příspěvků z oblíbených blogů a wiki
- zobrazení oblíbených *web site*
- zobrazení úkolů ze všech týmových *web site*, na kterých je uživatel členem

V ukázce kódu uvádím pouze kód pro automatické umístění web part s nejnovějšími příspěvky z oblíbených blogů. Kód pro ostatní web parts je obdobný.

5.3.2. Site definition pro blogy

Pro *web site* typu blog jsem upravil definici Blog, která je dostupná v systému SharePoint. Upravená šablona nese název Tieto Blog. Definice Blog šablony systému SharePoint je umístěna v TEMPLATE\SiteTemplates\Blog ve složce, kde je nainstalován SharePoint. Pro úpravu jsem zkopíroval obsah souboru onet.xml této šablony, vložil do svého projektu a provedl potřebné změny. Spolu se souborem onet.xml je potřeba zkopírovat a vložit do projektu i všechny soubory, které se k šabloně vztahují a jsou nasazovány pomocí modulů.

Úprava šablony se týká změny master šablony a umístění vlastního web part, který umožní označit web site jako oblíbenou. Definování master page i umístění web part je provedeno stejným způsobem jako v případě šablony pro publikační site, nebudu proto uvádět ukázky kódu.

5.3.3. Site definition pro WIKI

Zde je situace obdobná jako v případě blog *web site*. Pro své řešení jsem použil jíž hotovou *site definition* v systému SharePoint a provedl jsem v ní určité úpravy (především změna *master page*).

5.3.4. Site definition pro root web site blogů a WIKI

V intranetu budou všechny WIKI a blog *web site* umístěny pod jednou společnou *web site*, aby je uživatel mohl lehce najít a procházet. Domovské *web site* blogů i WIKI budou vytvořeny podle stejné šablony. V rámci této šablony jsem stejně jako v případě publikační *web site* musel vytvořit soubory webTemp, onet.xml a default.aspx. Vytváření blogů a WIKI pod jednou společnou *web site* probírám dále v kapitole *5.7. Vytváření blogů a WIKI web site dostupné všem uživatelům*.

Úvodní stránka pro tuto šablonu obsahuje dvě *web parts*. Jedna z nich slouží k zobrazení *web sites*, které v hierarchii leží o jednu úroveň níže, tedy blogy respektive WIKI. Druhý *web part* bude zobrazovat nejnovější příspěvky aktuálně označeného blogu nebo WIKI. Při označení vybrání nějaké *web site* v prvním *web part* se musí adekvátně aktualizovat obsah druhého web part. Jistě by se dalo navrhnout velké množství řešení pro propojení dvou *web part*. Já jsem se rozhodl tyto dvě komponenty implementovat jakožto tzv. *connectable web parts*, což je cesta podporovaná frameworkem systému SharePoint. Její výhoda spočívá v tom, že SharePoint je schopen takto vytvořené *web parts* rozpoznat a nabízet v uživatelském rozhraní některé speciální možnosti. Konkrétní implementace *connectable web*

parts bude popsána později, uvádím ukázku kódu pro vytvoření spojení mezi *web parts*, která je umístěna v souboru onet.xml.

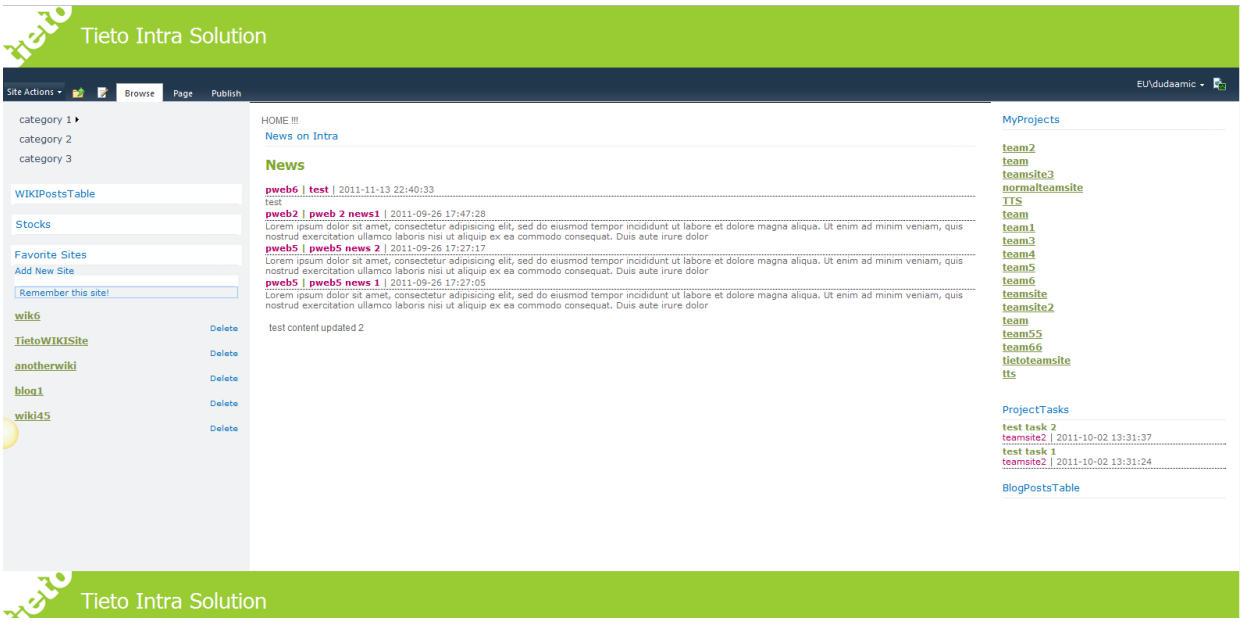
```
<AllUsersWebPart ID="SitesProvider" WebPartZoneID="MainZone" WebPartOrder="1">
 (zkráceno)</AllUsersWebPart>

<AllUsersWebPart ID="BPConsumer" WebPartZoneID="RightZone" WebPartOrder="1">
(zkráceno)</AllUsersWebPart>

<WebPartConnection ID="testConnection"
                        ProviderID="SitesProvider"
                        ProviderConnectionPointID="SubSitesProvider"
                        ConsumerID="BPConsumer"
                        ConsumerConnectionPointID="BlogPostsConsumer" />
```

Nejdříve jsou obě komponenty specifikovány pomocí elementu AllUsersWebPart stejně jako v předchozích případech, proto jsem část kódu vynechal. V element WebPartConnection je definováno spojení mezi *web parts*. Atribut ProviderID je shodný s atributem ID elementu AllUsersWebPart a znamená, že tento *web part* bude v roli poskytovatele dat. V Atributu ConsumerID je naopak definuje *web part*, který data přijímá. Atribute ProviderConnectionPointID a ConsumerConnectionPointID definují ID bodů připojení u poskytovatele respektive spotřebitele dat [1].

5.4. Implementace master pages

Pro vlastní site definice jsem implementoval také vlastní master page. Master page pro jednotlivé typy *web site* se liší v detailech, všechny ale odpovídají jednotnému schématu barev a mají stejné rozložení částí stránky.

Na následujícím obrázku (Obrázek 10) uvádím ukázku Master Page pro publikační *web site*. Na úvodní stránce je vidět několik rozmístěných *web parts*, například pro zobrazení novinek ze všech publikačních *web site*, na které má uživatel přístup (uprostřed), zobrazení oblíbených *web site* (v levém sloupci dole) nebo zobrazení teamových *web site*, na kterých je uživatel členem (v pravém sloupci nahoře).



Obrázek 10 Ukázka Master Page pro publikační web site

5.4.2. Nasazení *master page*

Master page je nasazována do seznamu _catalogs/masterpage. Jedná se o seznam, který je automaticky vytvořen na každé web site a slouží k uložení Master Pages a také šablon stránek.

Pro nasazení souboru je potřeba vytvořit XML definici modulu, který bude zpracován v rámci nějaké vytvořené funkcionality (*feature*). Definice modulu má následující podobu:

```
<Module Name="MasterPageModule">
       <File Path="MasterPageModule\_catalogs\masterpage\publish.master"
       Url="_catalogs/masterpage/publish.master" />
</Module>
```

Modulem je specifikováno, kam se má určitý soubor (cesta specifikována atributem Path) nahrát ve struktuře sytému SharePoint (atribut Url).

5.4.1. Nastylování *master page*

Soubor s kaskádovými styly nasazuji pomocí *feature* pro oblast site collection do seznamu Style Library. Jedná se o systémový seznam, který je v systému SharePoint automaticky vytvořen. Do stejného seznamu nasazuji také potřebné obrázky. Soubory, které se nasazují, specifikuji modulem podobně jako v případě *master page*. Samotné vytvoření CSS stylů nebudu popisovat, není podstatné vzhledem k tématu diplomové práce.

5.4.2. Komponenty systému SharePoint použité na Master Page

Na Master Page je potřeba využít některé komponenty uživatelského rozhraní systému SharePoint, aby uživatelům zůstala přístupná určitá standardní funkcionalita.

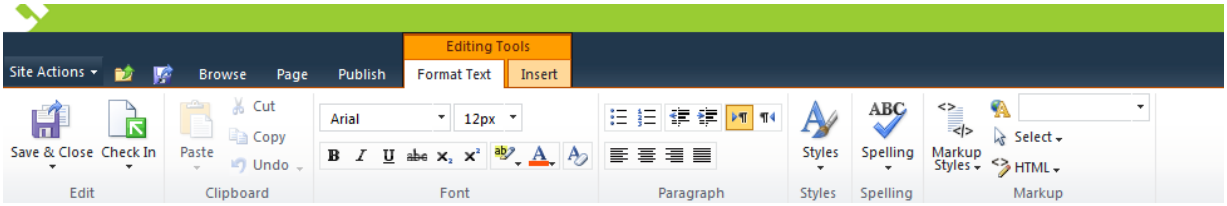
Pro standardní CSS styly a JavaScript systému SharePoint je potřeba do head sekce Master šablony přidat následující komponenty:

```
<SharePoint:CssLink runat="server" Version="4" />
<SharePoint:Theme runat="server" />
```

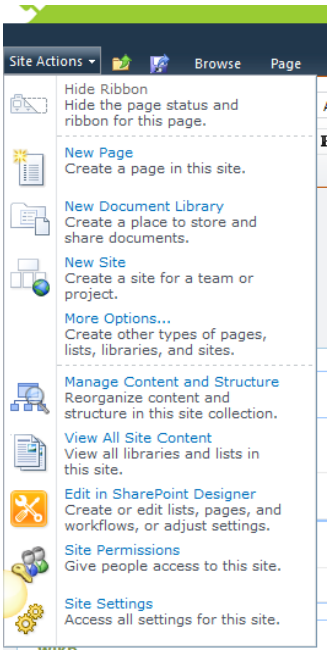
Komponenta SPPageManager, uvedená také v head sekci, slouží k řízení příkazů z a do standardního Ribbon panel [15]:

```
<SharePoint:SPPageManager runat="server" />
```

Dále v Master Page šabloně využívám komponentu SPRibbon (viz Obrázek 11). Komponenta SPRibbon je poměrně složitá, skládá se z několika částí, které dále obsahují některé další komponenty. Například komponentu SiteActions, která zajišťuje zobrazení hlavního menu v systému SharePoint (viz Obrázek 12). Při vkládání SPRibbon komponenty do vlastních Master page jsem využil jednak kódů hotových systémových Master Pages v systému SharePoint a také open-source projektu *Starter Master Pages for SharePoint 2010* [16]. Tento projekt obsahuje několik připravených Master Page, jejichž kódy jsou dobře okomentovány a jsou tak velmi užitečné, pokud programátor nemá ještě mnoho zkušeností s vytvářením vlastních Master Pages pro systém SharePoint.



Obrázek 11 Ukázka standardní SPRibbon komponenty



Obrázek 12 Menu systému SharePoint, které zobrazuje komponenta SiteActions

5.5. Šablony stránek

Šablony stránek představují předpis pro konkrétní stránky vytvořené v systému SharePoint a uložené v *content* databázi. Jejich smysl je jiný než v případě *master page*. Šablony stránek jsou samy závislé na určité *master page*, jejíž podobu dále rozšiřují o různé prvky, které nemohou být zasazeny přímo do *master page*, například o WYSIWYG editor v určité části stránky nebo *web part* zónu pro vložení libovolných prvků uživatelského rozhraní. V momentě, kdy uživatel vytvoří v systému novou *aspx* stránku, které bude uložena v databázi, je tato stránka odvozena od určité šablony.

Vytvořil jsem celkem tři šablony stránek, které korespondují s třísloupcovým designem *master page*. První šablona slouží k vytvoření čistě textové informativní stránky a obsahuje pouze WYSIWYG editor, který je umístěn ve středním sloupci. Druhá šablona obsahuje WYSIWYG editory ve všech třech sloupcích. Poslední neobsahuje žádné přednastavené prvky uživatelského rozhraní, obsahuje pouze zóny pro dosazení libovolných *web parts* ve všech třech sloupcích.

5.5.1. Nasazení šablon stránek

Nasazení šablon jsem vytvořil feature, která musí obsahovat definici modulu se specifikovanými soubory šablon. Jedná se o stejný postup jako v případě specifikace výchozí stránky u šablon *web site* (viz kapitola 5.3.1.2. onet.xml). V XML definici se tedy použije element file. Šablony se umísťují do speciálního seznamu *_catalogs/masterpage*.

Ukázka:

```
<File Path="MasterPageModule\PageTemplate3.aspx" Url="PageTemplate3.aspx"
Type="GhostableInLibrary">
      <Property Name="Title" Value="Tieto Page Template 3" />
      <Property Name="ContentType"
Value="$Resources:cmscore,contenttype_pagelayout_name;" />
      <Property Name="PublishingPreviewImage"
Value="~SiteCollection/_catalogs/masterpage/$Resources:core,Culture;/Preview
Images/ArticleLinks.png,
~SiteCollection/_catalogs/masterpage/$Resources:core,Culture;/Preview
Images/ArticleLinks.png" />
      <Property Name="PublishingAssociatedContentType"
Value=";#Page;#0x010100C568DB52D9D0A14D9B2FDCC96666E9F2007948130EC3DB064584E219954237AF39
;#" />
    </File>
```

V definici je dobré si povšimnout vlastnosti *ContentType* nastavené na speciální hodnotu určující, že se jedná o šablonu stránek (přesná hodnota je získána ze zdrojů systému pomocí zápisu *$Resources: ...*). Dále je vidět nastavení cesty k obrázku, která slouží jako ikona pro danou šablonu, a nakonec nastavení *content* typu, se kterým může být daná šablona použita. Protože tato šablona má takto definovaný *content* type *Page*, může být použita pouze, bude-li stránka vytvářena jakožto typ *Page* nebo typ, který dědí z *Page.*

Při vývoji šablon jsem narazil na problém s opětovným nasazováním upravené šablony [18], což je samozřejmě při vývoji velmi častá věc. SharePoint odmítne při aktivaci funkcionality již existující šablonu přepsat. Řešením by bylo danou šablonu nejdříve smazat například pomocí programu *SharePoint Desinger.* Pokud už ale je z této šablony odvozena nějaká stránka, není ani toto smazání možné. Proto je nutné aktualizaci již existující šablony vyřešit programově při události *FeatureActivated*, přičemž postup je následující:

1. Na aktualizované šabloně je potřeba provést programově *CheckOut* přes API systému SharePoint
2. Přečíst nový soubor z disku do *Stream* nějakou standardní metodo
3. Nahradit obsah starého souobru a provést *CheckIn*.

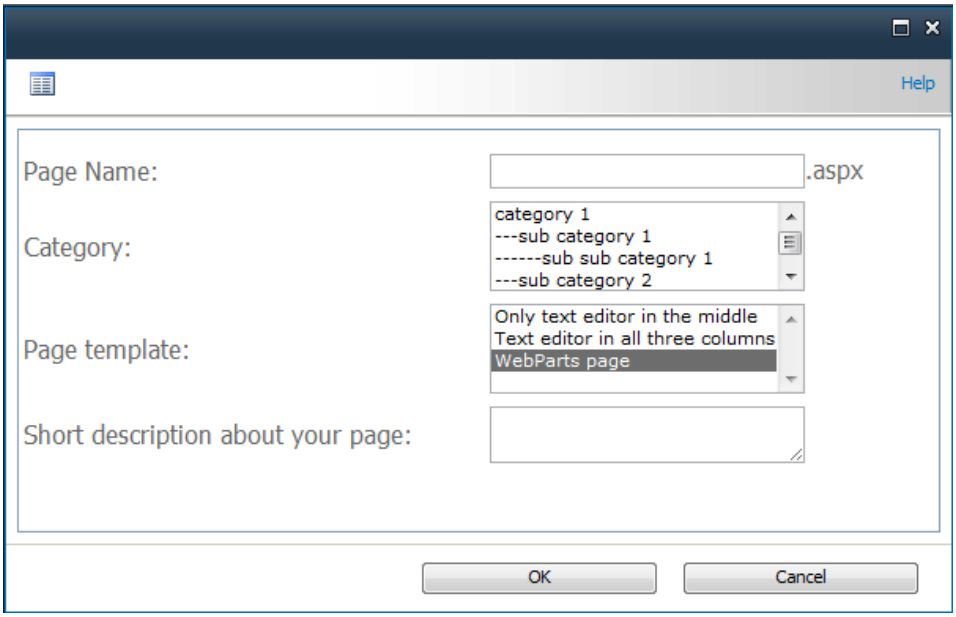
Protože v rámci tohoto modulu více šablon zobecnil jsem celý postup takto:

1. Přečtení XML souboru, ve kterém je definován modul s šablonami
2. Iterace přes všechny File elementy
3. Provedení nahrazení staré šablony za novou podle předchozích tří bodů.

5.5.2. Programové vytvoření stránky podle šablony

Vytváření stránek podle určité šablony je funkcionalita běžně dostupná v systému SharePoint, jenže jedním z požadavků byla kategorizace stránek. Vybrání kategorie SharePoint sám neumožní, i když seznam stránek obsahuje sloupec pro spojení s kategorií. Proto jsem musel vytvořit speciální rozhraní pro vytváření stránek a jejich přiřazení do kategorií. Pro tento účel jsem rozšířil standardní menu systému SharePoint o novou položku *New page*, která je dostupná pouze administrátorům publikační *web site*. Postup rozšíření menu je blíže popsán v kapitole *5.7.1. Prvky uživatelského rozhraní*. Dále jsem musel vytvořit aplikační stránku, která bude zobrazena v rámci dialogu pro vytvoření nové publikační stránky. Na aplikační stránce jsem vytvořil rozhraní, které umožní uživateli zvolit název, šablonu, kategorii a popis nově vytvářené publikační stránky.

Po kliknutí na novou položku v menu se uživateli zobrazí následující dialogové okno:



Obrázek 13 Dialogové okno pro vytvoření nové stránky

Pro samotné programové vytvoření stránky je potřeba využít knihovnu *Microsoft.SharePoint.Publishing*. Postup popíši spolu s ukázkou části kódu.

Nejdříve se vytvoří objekt typu *PublishingWeb* na základě aktuálního *SPWeb* objektu (aktuální *web site*). Dále je potřeba vyhledat potřebnou šablonu stránky podle jména.

```
PublishingWeb publishingWeb = PublishingWeb.GetPublishingWeb(SharePointContext.Web);
List<PageLayout> layouts = new List<PageLayout>(publishingWeb.GetAvailablePageLayouts());
PageLayout pageLayout = layouts.Find(
     delegate(PageLayout l)
     {
        return l.Name.Equals(pageLayoutName, StringComparison.CurrentCultureIgnoreCase);
     }
 );
```

Pokud je šablona stránky nalezena, je potřeba přidat novou položku do kolekce stránek.

```
PublishingPageCollection publishingPageCollection = publishingWeb.GetPublishingPages();
SharePointContext.Web.AllowUnsafeUpdates = true;
PublishingPage newPage = null;
newPage = publishingPageCollection.Add(pageName + ".aspx", pageLayout);
```

Na nově vytvořené stránce se dále nastaví potřebné vlastnosti (v ukázce uvádím pouze nastavení kategorie) a poté je potřeba zavolat metody *Update* a *CheckIn*. Metoda *CheckIn* vyžaduje popis pro historii dokumentu.

```
newPage.ListItem["PCategory"] = new SPFieldLookupValue(categoryId, category.Title);
newPage.Update();
newPage.CheckIn("category setted");
```

5.5.3. Zobrazení kategorizovaných stránek

Pro zobrazení kategorií jsem využil komponentu *AspMenu* (standardně dostupná v knihovnách systému SharePoint), která slouží k zobrazení navigace v hiearchické struktuře. Tuto komponentu jsem vložil do *master* šablony pro publikační web do levého sloupce nahoru (viz obrázek 10). Aby komponenta mohla zobrazovat uložené kategorie musel jsem **naimplementovat třídu dědicí z abstraktní třídy SiteMapProvider**. Tato třída načítá informace o kategoriích z doménové vrstvy a poskytuje je komponentě v takovém tvaru, aby korektně zobrazila strom kategorií se správnými odkazy. Nový *site map provider* jsem dále musel zaregistrovat v souboru web.config.

Odkazy z tohoto menu směřují na implementovanou aplikační stránku, která zobrazí seznam stránek potřících do dané kategorie. Správná kategorie je rozpoznána podle paramteru v URL adrese, načtení stránek samozřejmě probíhá přes třídu IPagesService, která je umístěna v doménové vrstvě.

5.6. Agregace dat z oblíbených blogů a WIKI site

Jednou z rozšiřujících funkcionalit je agregování dat z oblíbených blogů a WIKI, aby je bylo možno uživateli rychle zobrazit třeba na úvodní stránce intranetu (nebo kdekoliv jinde, podle uvážení administrátorů intranetu). Agregovaná data se ukládají do listů AggregatedBlogPosts a AggregatedWIKIPosts, které jsou umístěny na domovské *web site* intranetu, jak bylo popsáno dříve v kapitole zabývající se uložením dat. Pro agregaci dat jsem vytvořil vlastní *Timer Job* pro systém SharePoint. *Timer Job* představuje funkcionalitu, která se spouští v časových intervalech. Tyto funkcionality jsou spouštěny pomocí speciální Windows služby SharePoint 2010 Timer.

5.6.1. Timer Job pro systém SharePoint a jeho vytvoření

Vlastní Timer Job je tvořen třídou, která musí dědit z bázové třídy SPJobDefinition. Pro implementaci vlastní funkcionality je potřeba přepsat virtuální metodu Execute této bázové třídy. Job je přidán do systému při nasazení řešení, nicméně pro jeho správné fungování je nutné, aby byla korektně nastavena doba spouštění a případně další vlastnosti. Nastavení provádím pomocí *Event Reciever* funkcionality (*feature*), která je aktivována při nasazování řešení. V metodě FeatureActivated se provede nastavení periody spouštění.

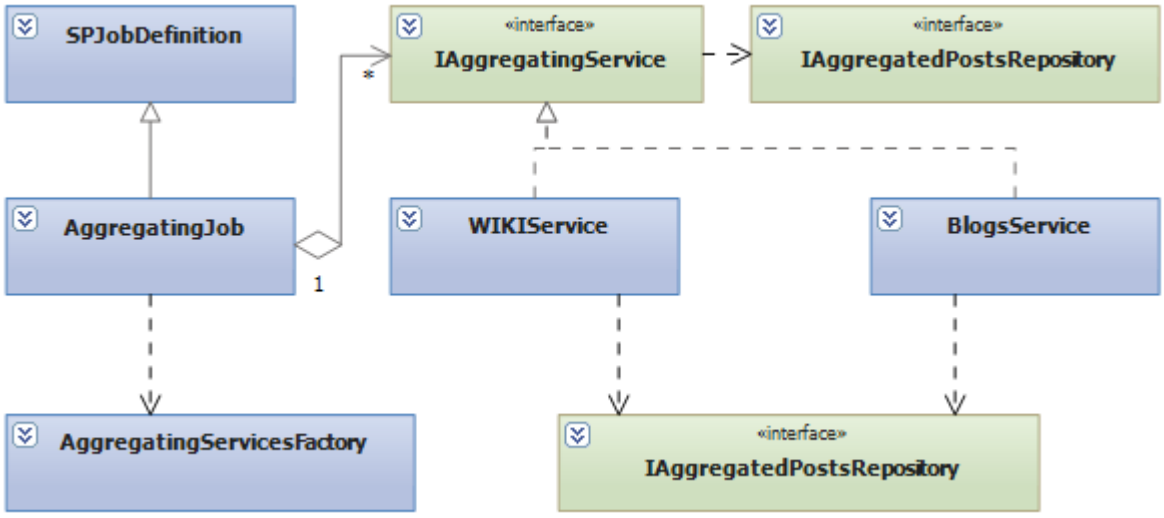
Ukázka části kódu:

```
public override void FeatureActivated(SPFeatureReceiverProperties properties)
{
     SPWebApplication webApp = site.WebApplication;

     string name = "Tieto aggregating job";

     foreach (SPJobDefinition job in webApp.JobDefinitions)
         if (job.Name == name) job.Delete();

     AggregatingJob agrJob = new AggregatingJob(webApp, name);
     SPDailySchedule schedule = new SPDailySchedule();
     schedule.BeginHour = 1;
     schedule.EndHour = 2;

     agrJob.Schedule = schedule;
     agrJob.Update();
}
```

Pro *Timer Job* jsem definoval jméno *Tieto aggregating job*. Nejprve se provádí kontrola, zda Job s tímto jménem již v systému existuje (například z předchozího nasazení řešení). Pokud ano, je odstraněn. Dále je instanciován samotný *Job* a pomocí třídy SPDailySchedule je nastaveno, aby se spouštěl jedenkrát denně mezi jednou a druhou hodinou ranní. Metoda Update zařídí uložení stavu do systému a propagování nastavení na všechny počítače v SharePoint farmě.

5.6.2. Nastavení přes *Unity* framework

Logika získávání dat z blogů s WIKI není umístěna ve třídě mého *Timer Job*, ale v třídách umístěných na vrstvě doménové logiky. Je potřebné, aby bylo možné agregování rozšířit i na další typy dat. Proto jsem implementaci navrhl tak, že *Timer Job* pracuje s polem objektů, které implementují obecné rozhraní IAggregatingInterface. Toto rozhraní dále implementuji ve třídách, které provádějí agregaci příspěvků z blogů respektive WIKI. Pole objektů, které provádějí agregaci, je získáno od pomocné *factory* třídy, která jej vytvoří na základě konfigurace *Unity* frameworku. Díky tomuto přístupu může být například snadno vytvořena nová knihovna zajišťující agregaci jiného typu dat (například novinek) a pouhou úpravou konfigurace *Unity* přidána ke zpracovávání pomoci *Timer Job*. Princip bude jasnější z následujícího třídního diagramu:



Obrázek 14 Class diagram tříd, které se používají pro agregování dat

Naplnění objektů typu IAggregationService dosaženo následující konfigurací *Unity* frameworku:

```
<register type="MOSK.Intranet.Core.Domain.Seedwork.AggregatingServicesFactory"
mapTo="MOSK.Intranet.Core.Domain.Seedwork.AggregatingServicesFactory">
   <property name="Services">
     <array>
       <dependency type="MOSK.Intranet.Core.Domain.Blogs.BlogsService" />
       <dependency type="MOSK.Intranet.Core.Domain.Wiki.WikiService" />
     </array>
   </property>
</register>
```

Toto nastavení zaručí inicializace pole Services v instance třídy AggregatingServicesFactory. Z tohoto objektu jej dále získá objekt AggregatingJob. Toto pole nemůžu inicializovat přímo ve třídě AggregatingJob, protože instance úlohy je vytvořena a spuštěna systémem SharePoint, nevytváří se přes *Unity* framework.

Při použití *Unity* frameworku ve spojení s *Timer Job* systému SharePoint nastává ještě problém s načtením konfigurace. *Timer Job* je spuštěn jiným způsobem než webové aplikace (běží v rámci procesu OWSTIMER.EXE) a proto nastává problém, jakým způsobem načíst nastavení, která jsou uložena v souboru web.config. K načtení je potřeba použít třídu WebConfigurationManager a při otevírání konfigurace specifikovat jméno webové aplikace. Třída SPJobDefinition obsahuje vlastnost WebApplication a tedy mohu k této vlastnosti přistupovat i ve třídě AgrgegatingJob (webová aplikace byla specifikována při vytváření *Timer Job*, viz předchozí ukázka kódu v kapitole *5.6.1 . Timer Job pro systém SharePoint a jeho vytvoření*).

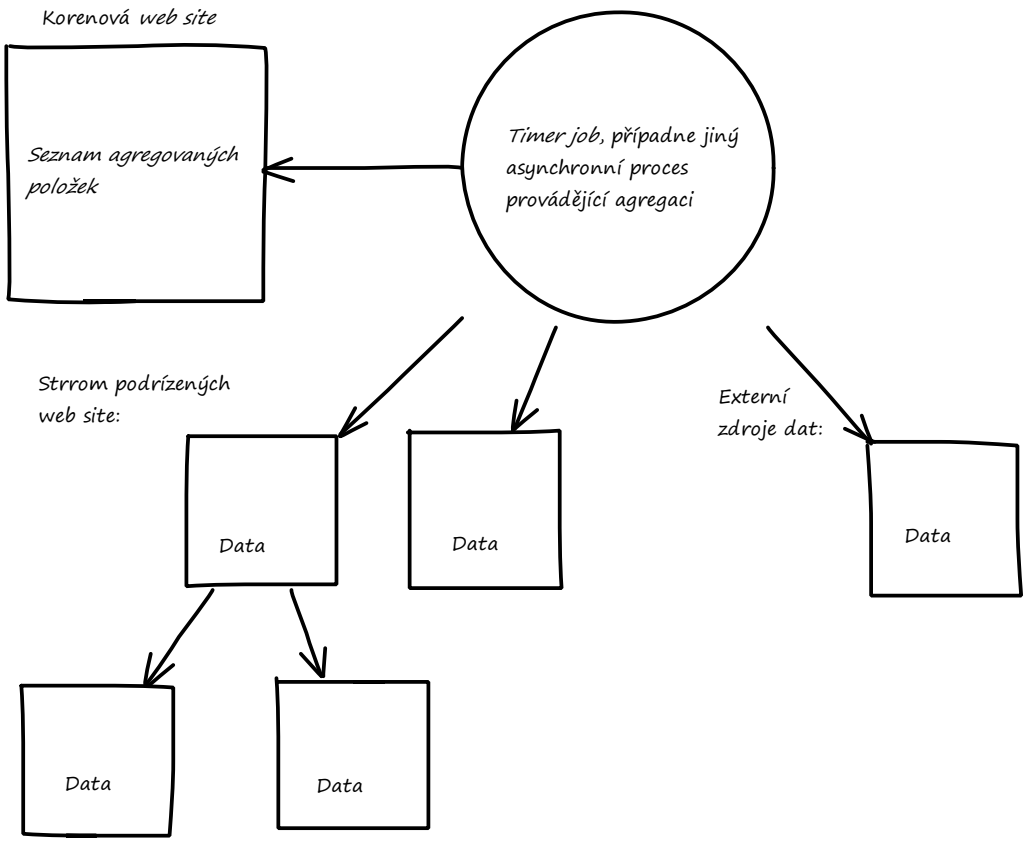
Ukázka kódu načtení *Unity* konfiguratce v rámci *Timer Job*:

```
Configuration config = WebConfigurationManager.OpenWebConfiguration("/",
WebApplication.Name);
var unitySection = (UnityConfigurationSection)config.GetSection("unity");
```

5.6.3. Provedení agregace dat

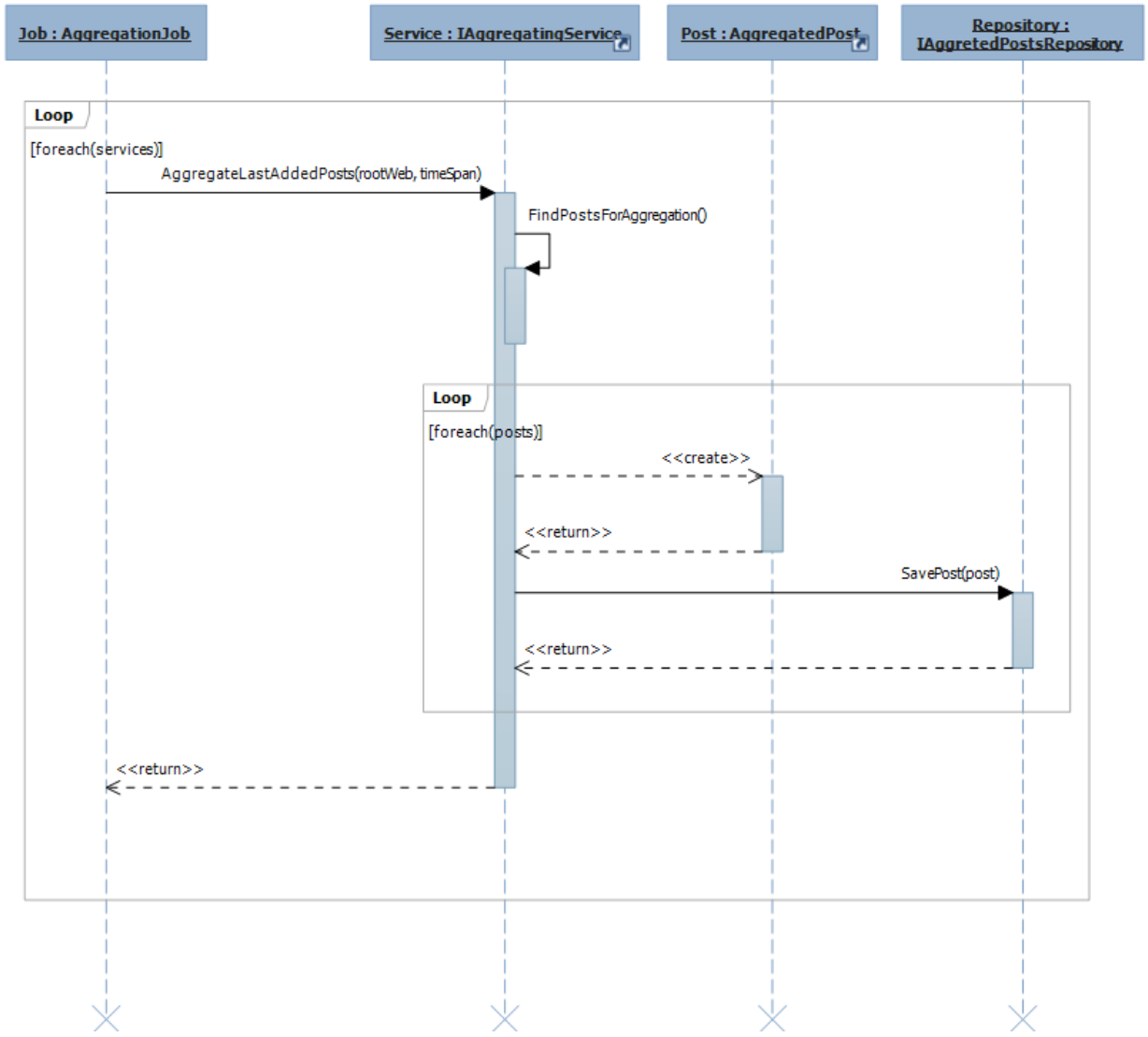
Pro agregování dat je potřeba projít všechny blog a wiki *web site*, které má uživatel uloženy jako oblíbené a některé informace, uložené v seznamech těchto *web site*, zkopírovat do agregačních seznamů. Tento princip je popsán v knize Design Solutions for Microsoft SharePoint 2010 [6] pod názvem „Union List Aggregation“ a znázorněn následujícím obrázkem (Obrázek 15).

Ve svém řešení sleduji tento vzor. Timer Job spustí metodu pro agregování dat nad každou servisní třídou.



Obrázek 15 "Union-aggregated list pattern" [6]

Následující sekvenční diagram zobrazuje průběh provedení agregace (Obrázek 16):



Obrázek 16 Sekvenční diagram práce agregačního Timer Jobu

5.6.4. Agregace dat do relační databáze

Z důvodu rychlejšího načítání dat bývá často pro agregaci dat použita relační databáze, proto jsem tento způsob implementoval i ve své práci. Struktura tabulek v databázi jsem navrhl univerzálněji než strukturu seznamů. V systému SharePoint jsem příspěvky blogů a WIKI ukládal do samostatných seznamů z důvodu rychlého načítání. V relační databázi používám pro uložení příspěvků z blogů i WIKI jednu tabulku, do které mohou být případně uložena i jiná agregovaná data. O jaká data se jedná (zda jde o

příspěvek z blogu, WIKI nebo něco jiného) rozpoznám podle informací o *web site*, ke které se příspěvek vztahuje. Databáze je vytvořena v MS SQL Server 2008.

5.6.4.1. Tabulky v relační databázi

Celkově jsem pro uložení dat vytvořil tři tabulky:

1. Tabulka AggregatedPosts slouží k uložení agregovaných dat.

2. Tabulku WebSites do které se uloží informace o *web site*, kterou některý uživatel označil jako oblíbenou. **Mezi touto tabulkou a tabulkou AggregatedPosts je vazba 1:N.** K jednomu záznamu v tabulce WebSites se vztahuje více záznamů v tabulce AggregatedPosts.

3. Tabulka UserFavoriteWebSites, která slouží k uložení vazby uživatele na určitou *web site*. **Tuto tabulku si lze představit jako tabulka zajišťujcí relaci M:N mezi uživateli a tabulkou WebSites.** Nicméně uživatelé nejsou uložení v rámci relační databáze ale systému SharePoint. Mezi tabulkou WebSites a UserFavoriteWebSites je tedy vazba 1:N, více uživatelů může mít oblíbenou stejnou *web site*.

5.6.4.2. Implementace ukládání do databáze

Rozhraní IAggreationService, které jsem navrhl jakožto společné rozhraní pro všechny třídy, které provádějí agregaci, předpokládá pro ukládání dat třídu implementující rozhraní IAggregatedPostsRepository (viz Obrázek 14). Abych mohl ukládat data do relační databáze vytvořil jsem další implementaci tohoto rozhraní. Stejně tak jsem potřeboval naimplementovat rozhraní IFavoriteSitesRepository, aby bylo možné ukládat uživatelem zvolené oblíbené *web site* do relační databáze. Tyto nové implementace jsem umístil do samostatné *assembly* Tieto.Intranet.DBAccessLayer. Kam budou data nakonec uložena, závisí opět pouze na nastavení *Unity* frameworku, viz následující ukázka.

Nejprve je v konfiguraci potřeba specifikovat použití nové assembly:

```
<assembly name=" Tieto.Intranet.DBAccessLayer, version=1.0.0.0, culture=neutral,
publickeytoken=43b58233b23470e0" />
```

A dale změnit mapování rozhraní dvou výše uvedených repository tříd:

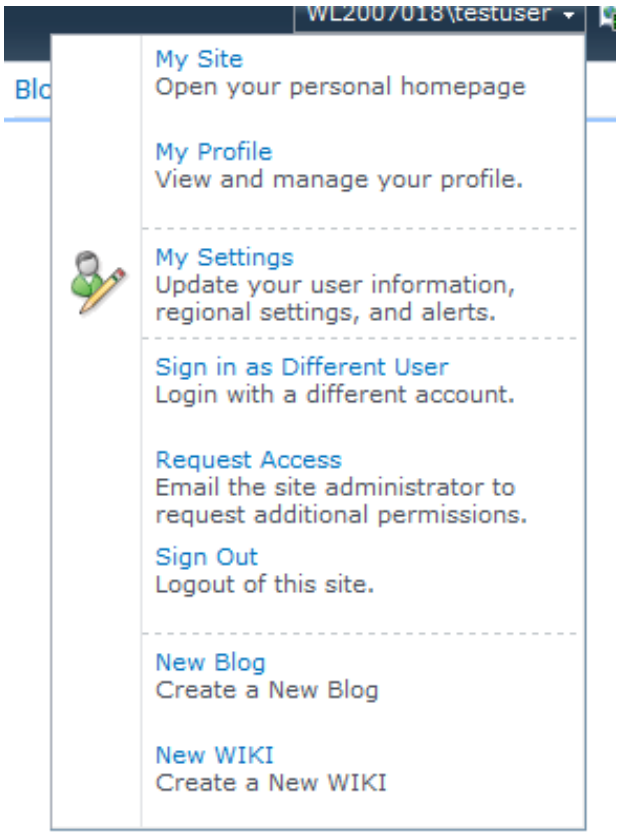
```
<register
type="Tieto.Intranet.Core.Domain.Aggregation.Repositories.IFavoriteSitesRepository"
mapTo="Tieto.Intranet.DBAccessLayer.Repositories.FavoritesSitesRepository">
</register>

<register
type="Tieto.Intranet.Core.Domain.Aggregation.Repositories.IAggregatedPostsRepository"
mapTo=" Tieto.Intranet.DBAccessLayer.Repositories.AggregatingRepository">
</register>
```

5.7. Vytváření blogů a WIKI web site dostupné všem uživatelům

Vytváření nových *web site* ve výchozím režimu probíhá přes dialogové okno Create, kde uživatel volí, co chce vytvořit (v tomto případě tedy novou *web site*) a dále podle které šablony. Tato funkcionalita ale obvykle nebývá povolena všem uživatelům intranetu. Proto jeden z požadavků byl rozšíření funkcionality systému o zpřístupnění možnosti vytvoření těchto *web site* každému uživateli.

5.7.1. Prvky uživatelského rozhraní

Do standardního menu systému SharePoint, které se nachází vpravo na horním panelu (viz Obrázek 17), jsem umístil dvě nové položky, New Blog a New WIKI. Výslednou podobu menu je vidět na obrázku níže:



Obrázek 17 Standardní menu rozšířené o položky New Blog a New WIKI

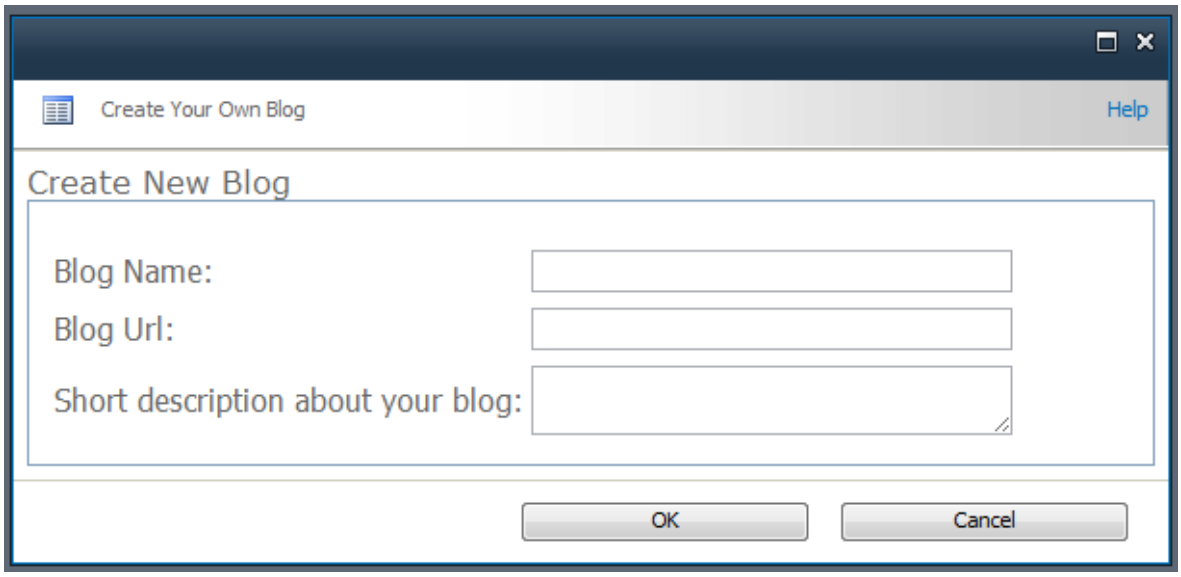
Rozšíření menu jsem provedl deklarativním způsobem pomocí *feature* nazvané MenuExtensions, která se aktivuje na oblast site collection. V elementech této feature (v souboru elements.xml) jsem použil prvek

CustomAction, pomocí nějž se dá provést rozšíření standardních ovládacích prvků, například i Ribbon komponenty.

Ukázka kódu:

```
<CustomAction
  Id="newBlogAction"
  GroupId="PersonalActions"
  Location="Microsoft.SharePoint.StandardMenu"
  Sequence="1000"
  Title="New Blog"
  Description="Create a New Blog"
  ImageUrl="_layouts/1033/images/myimage.png">

       <UrlAction Url="javascript:OpenPopUpPage('{SiteUrl}/_layouts/Pages/NewBlog.aspx',
null, 600, 250);" />

  </CustomAction>
```

V ukázce kódu je vidět specifikované ID, titulek a popis nové akce. Nejzajímavějším atributem je Location, který určuje kam se má tato položka umístit. Pro umístění do pravého standardního menu slouží hodnota *Microsoft.SharePoint.StandardMenu* [1]. Element UrlAction určuje, že položka v menu bude představovat odkaz na nějakou adresu. Atribut Url neobsahuje pouze adresu, ale také volání JavaScriptové funkce OpenPopUpPage. Jedná se o funkci obsaženou v systému SharePoint, která zajistí zobrazení dialogového okna pro určitou URL, která odpovídá grafickou podobou standardním dialogovým oknům systému SharePoint. Výraz {SiteUrl} nahradí SharePoint automaticky adresou aktuální web site. Stránka NewBlog.aspx je vytvořena jako aplikační stránka. Výsledná podoba dialogu je vidět na obrázku níže. Vzhled dialogu je stejný v obou případech dialogů.



Obrázek 18 Dialog pro vytvoření vlastního blogu

U aplikačních stránek, které jsou použity v dialozích, používám master page dialog.master, která je obsažena v systému SharePoint. Tato master page obsahuje některé prvky, které jsou typické pro dialogy v systému SharePoint (například tlačítka OK a Cancel, horní popisek dialogu, odkaz Help).

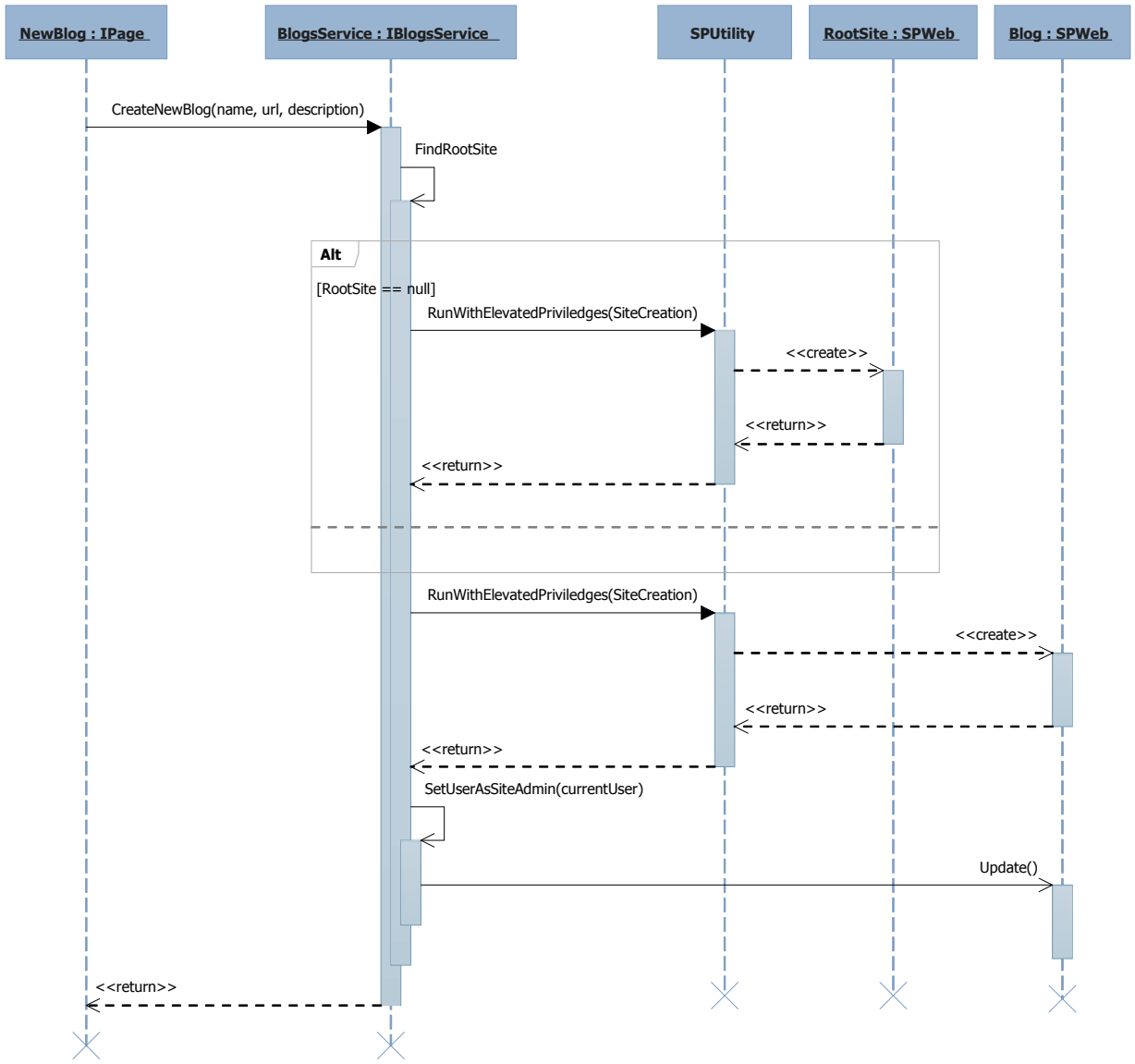
5.7.2. Logika vytváření blog a WIKI web site

Po kliknutí na tlačítko OK v dialogu (viz Obrázek 18) se provede kontrola, zda uživatel vyplnil požadované údaje korektně a zda již neexistuje web site s požadovanou adresou. Pakliže tyto kontroly proběhnou v pořádku, spustí se samotné vytváření nové web site.

Všechny blogy a WIKI chci umísťovat pod jednu společnou *web site*. V případě blogů to má být web site na adrese *adresa-intranetu*/BlogsRootSite a v případě WIKI *adresa-intranetu*/WikiRootSite. Tyto kořenové *web site* budou zobrazovat přehledným způsobem všechny dostupné blogy respektive WIKI a budou vytvořeny podle speciální šablony (viz kapitola *5.3.4. Site definition pro root web site blogů a WIKI*). V prvním kroku vytváření nového blogu nebo WIKI je tedy potřeba najít tuto kořenovou *web site*, případně ji vytvořit, jedná-li se o první blog nebo WIKI v intranetu. *(Poznámka: Tyto kořenové web site budou pravděpodobně vytvořeny programátory, kteří budou mít na starosti nasazení řešení. Nicméně je lepší ošetřit i variantu, že ještě neexistují.)*

Po nalezení nebo vytvoření kořenové *web site* samotné vytvoření blogu nebo WIKI provedu voláním metody Add kolekce Webs objektu SPWeb, který představuje kořenovou *web site*. Metodě Add jsou přidány všechny potřebné parametry jako název, url a popis. Metoda Add vrátí nový objekt typu SPWeb. Dále je nutné uživateli, který vytvořil blog, dát plnou kontrolu nad vytvořenou web site. Tato logika je samozřejmě umístěna v přístušných třídách doménové vrstvy.

Průběh vytváření web site je znázorněn na následujícím sekvenčním diagramu.



Obrázek 19 Sekvenční diagram znázorňující vytváření nového blogu

Při programovém vytváření nové web site nastává problém s právy aktuálního uživatele. Spouštěný kód, který nějakým způsobem ovlivňuje systém SharePoint, je implicitně omezován právy uživatele, který jeho spuštění vyvolal. Kód, který má být spouštěn s plnými právy bez ohledu na aktuálního uživatele, musí být spouštěn přes pomocnou třídu SPUtility a její statickou metodu RunWithElevatedPrivileges. Ta přijímá jako parametr delegáta odkazující na metodu, která má být spuštěna s plnými právy [1].

I když je kód spuštěn přes metodu RunWithElevatedPrivileges může vzniknout problém s právy, pokud se pracuje s objekty, které představují určité entity systému SharePoint (například SPWeb, SPSite) a byly vytvořeny mimo blok RunWithElevatedPrivileges. Proto je například v metodě CreateSite nutné znovu vytvořit objekt SPWeb představující kořenovou *web site* pro nově vznikající blog nebo WIKI, ne jej předávat metodě jako parametr. Problém bude jasnější z následující ukázky kódu:

```
protected void CreateSite(SPWeb rootWeb)
{
       SPSecurity.RunWithElevatedPrivileges(
                    new SPSecurity.CodeToRunElevated(() =>
                    {
                        SPWeb newWeb = rootWeb.Webs.Add( …
```

V tomto případě nastane problém s právy při volání metody Add pro vytvoření nové web site. Správný postup je následující:

```
protected void CreateSite(string rootWebUrl, Guid siteId)
{
       SPSecurity.RunWithElevatedPrivileges(
                    new SPSecurity.CodeToRunElevated(() =>
                    {
                       using (SPSite site = new SPSite(siteId))
                       {
                           using (SPWeb root = site.OpenWeb(rootWebUrl))
                           {
                              using (SPWeb newWeb = root.Webs.Add( …
```

V tomto případě problém s právy nenastane, protože objekt root typu SPWeb je vytvořen v rámci bloku RunWithElevatedPrivileges. *(Poznámka: Pro jednoduchost není uveden kód metody celý, jsou zjednodušeny také parametry metody.)*

Je dobré si povšimnout, že objekty typu SPSite nebo SPWeb jsou vytvářeny v rámci bloku *using*. Důvodem je, že tyto objekty na pozadí využívají neřízený (*unmanaged*) kód a paměť. Řízená část je mnohem menší než neřízená, a proto *garbage collector* neuvolňuje tyto objekty z paměti příliš rychle. Z tohoto důvodu není dobré ponechávat odstraňování objektů pomocí *garbage collector*. [17]

Uživateli, který vytvořil *web site*, je potřeba udělit práva pro plnou kontrolu. Nejdříve přeruším dědění práv nové z rodičovské *web site* pomocí metody BreakRoleInheritance nad objektem SPWeb.

```
newWeb.BreakRoleInheritance(true);
```

Následující kód již uděluje samotná práva uživateli (reprezentován objektem typu *SPUser*) pomocí tříd *SPRoleAssigment* a *SPRoleDefinition*. Uložení změn nad objektem *SPWeb* v rámci GET požadavku musí být povoleno vlastností *AllowUnsafeUpdates* [1]. Změny jsou nakonec uloženy metodou *Update*.

```
newWeb.AllowUnsafeUpdates = true;
SPUser elevatedContextUser = newWeb.EnsureUser(loginName);
SPRoleAssignment ownerRoleAssignment = new SPRoleAssignment(elevatedContextUser);
SPRoleDefinition fullControl = newWeb.RoleDefinitions[@"Full Control"];
ownerRoleAssignment.RoleDefinitionBindings.Add(fullControl);
newWeb.RoleAssignments.Add(ownerRoleAssignment);
newWeb.Update();
newWeb.AllowUnsafeUpdates = false;
```

5.8. Implementace logiky a uživatelského rozhraní pro načítání a zobrazení dat z veřejných zdrojů.

Implementoval jsem načítání dat z veřejného zdroje informací o aktuálních hodnotách akcií vybraných společností. Pro získání těchto údajů jsem využil veřejně dostupnou službu společnosti Yahoo!, službu Yahoo! Finance. Informace jsou dostupné přes klasické webové rozhraní na adrese http://download.finance.yahoo.com. Služba ale umožňuje i získání informací ve formátech, vhodných pro automatizované zpracování, například csv. Zasláním HTTP požadavku ve tvaru:

"*http://download.finance.yahoo.com/d/quotes.csv?s=GOOG,MSFT,CSCO&f=sl1c1d1&e=.csv*",
získám požadované informace ve formátu CSV.

5.8.1. Business Connectivity Services

Business Connectivity Services (dále jen BCS) jsou množinou funkcionalit, služeb a nástrojů, které umožňují vytvářet řešení v systému SharePoint integrovaná s jinými systémy a službami [2]. BCS jsou novou možností pro integraci s externími systémy existující od verze 2010, mající nahradit starší řešení Business Data Catalog, které existovalo ve verzi 2007 [1]. Integrace dat pomocí BCS přináší mnohé výhody, především co se týče další prezentace a použití dat v systému SharePoint.[2]

Při integraci systému jsou použity **externí content typy**, které jsou popisem externích datových zdrojů, jejich funkčnosti a chování v rámci SharePoint a Office systému. [2]

Pro integraci informací o akciích jsem také použil BCS, což mi například umožnilo zobrazit integrovaná data pomocí „*out of the box*“ prvků uživatelského rozhraní jako kterýkoliv jiný seznam v systému SharePoint.

Při připojování informací o akciích jakožto externího zdroje jsem musel definovat externí *content type*. Externí *content type* představují dva druhy tříd. Prvním druhem jsou třídy, představují entity z externího systému. V mém případě se jedná pouze o jednu entitu nazvanou Stock, která obsahuje vlastnosti pro název společnosti, poslední hodnotu akcií, procento změny a čas, kdy došlo ke změně. Druhým typem tříd jsou servisní třídy, které obsahují metody pro načtení a případně úpravu externích dat. V těchto třídách musí být implementovány některé specifické metody. V mém případě se jedná o třídu StocksService, u které ReadItem metoda vrací instaci třídy Stock podle určitého identifikátoru. Metoda ReadList vrací kolekci instanci třídy Stock. Tyto metody vytváří HTTP dotazy na výše zmíněnou službu

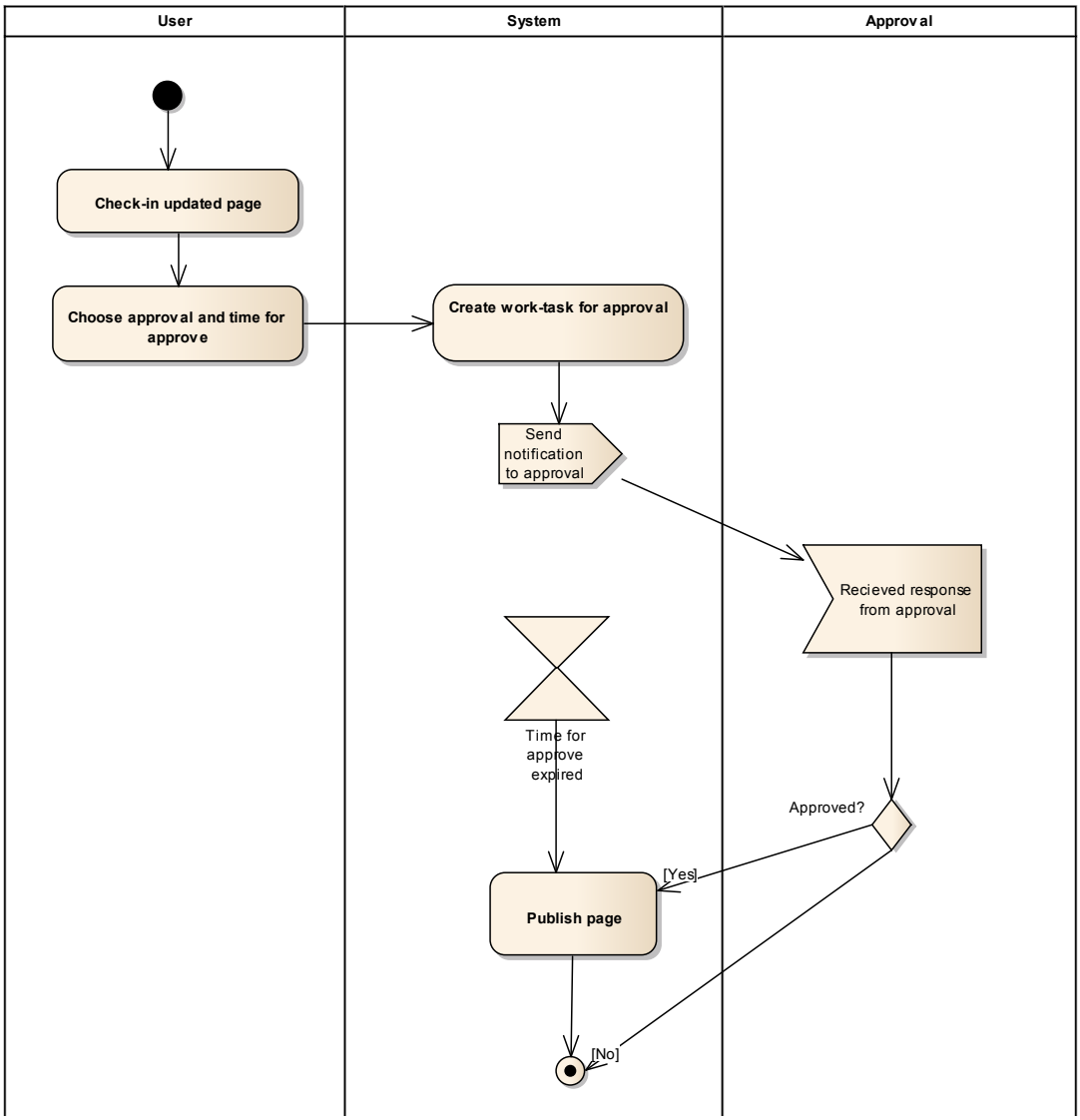
Yahoo! Finance, parsují CVS odpověď a mapují ji na objekty. Metody pro úpravu dat by v tomto případě samozřejmě nedávaly smysl, jejich implementace není na rozdíl od metod pro čtení dat nutná.

5.9. Vytvoření specifického workflow pro schválení a publikování obsahu / dokumentu

V systému SharePoint je „*out of the box*“ dostupné *workflow*, které může uživatel použít pro určení jiné osoby ke schválení jím vytvořeného dokumentu. Takto spuštěné *workflow* poté čeká na akci od uživatele, který byl určen pro schvalování, zda dokument potvrdí nebo zamítne. V případě, že uživatel vybraný ke schvalování na výzvu nereaguje, dokument zůstává nepotvrzen.

Pro účely intranetového řešení jsem vytvořil velmi podobné *workflow* sloužící ke schválení určitého dokumentu (například publikační stránky) s tím rozdílem, že uživatel může zadat čas a datum, kdy nejpozději má být dokument publikován, i když se k němu uživatel vybraný pro schvalování nevyjádří. Nejedná se tedy o schvalování ve smyslu, kdy uživatel potřebuje povolení k publikování, ale spíše o žádost o revizi dokumentu.

Průběh procesu revize a publikování jsem znázornil diagramem aktivit na obrázku 20.



Obrázek 20 Diagram aktivit popisující průběh workflow pro revizi a publikování dokumentu

5.9.1. *Workflows* v prostředí systému SharePoint

Systém SharePoint umožňuje rozšíření o vlastní *workflow*, které se implementují v Workflow Foundation frameworku verze 3.5. Knihovny SharePointu obsahují množství implementovaných akcí, pomocí kterých je možné zajistit časté operace nad entitami systému SharePoint.

Je možno vytvářet *workflow* dvou druhů, sekvenční a „*State-Machine*“. Sekvenční *workflow* představuje seřazené aktivity, které jsou postupně vykonávány. Běh může být řízen pomocí podmínek a cyklů. *State-Machine workflow* je oproti sekvenčnímu řízeno událostmi. *Workflow* pro revizi a publikování jsem vytvořil jako sekvenční *workflow*.

5.9.1.1. Nastartování *workflow* a uložení úkolu pro provedení kontroly dokumentu

Uživatel, který spouští *workflow*, musí mít možnost zvolit uživatele, který by měl provést kontrolu a také datum a čas kdy musí být dokument publikován. Pro tento účel jsem vytvořil aplikační stránku, která bude sloužit jako startovací stránka *workflow*. Tato stránka obsahuje *drop down box* naplněný uživateli, kterým je možné svěřit revizi dokumentu a dále *DateTimeControl*, což je komponenta která umožní zvolit datum a čas. Tato komponenta je součástí knihoven SharePoint. Stejně jako ke spoustě jiných rozšíření v systému SharePoint i *workflow* je připojen XML soubor definující jeho vlastnosti. Jednou z těchto vlastností je i specifikace startovací stránky.

Dále potřebuji vybranému uživateli přiřadit úkol kontroly dokumentu. Ke každému spuštěnému *workflow* se mohou vztahovat úkoly, které jsou uložené ve speciálním seznamu (WorkflowTasks). Úkoly jsou odvozeny od content typu Task, který obsahuje různá pole, mimo jiné také pole AssignedTo, které určuje, komu je úkol přiřazen. Já ale potřebuji navíc pole pro označení, zda byl nebo nebyl dokument schválen a pole pro případné zdůvodnění. Proto jsem musel seznam úkolů rozšířit o další typ položek (*content* typ), který je do něj možné uložit. Vytvořil jsem tedy:

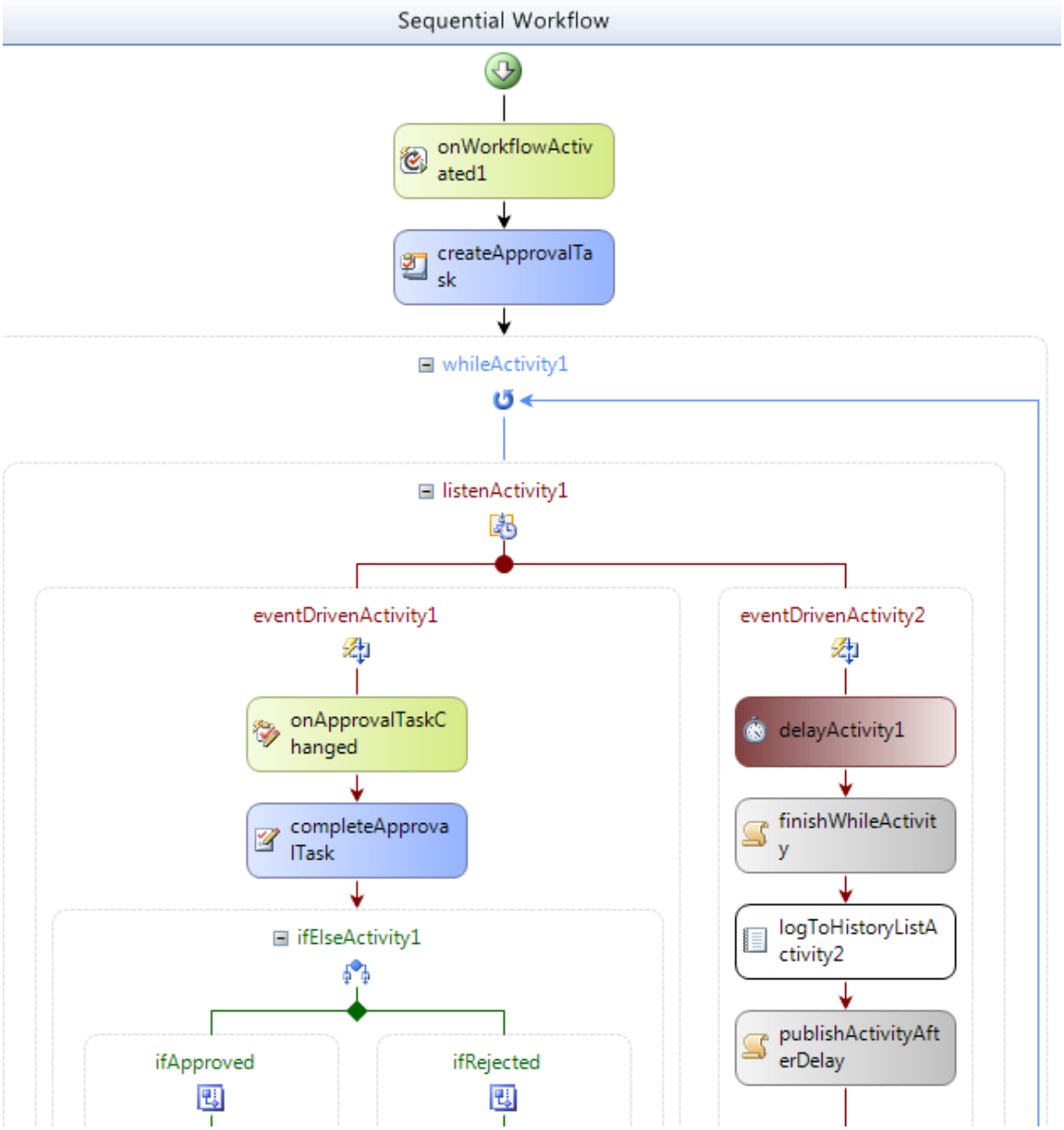
1. Definice dvou nových sloupců.
2. Definici nového *content* typu TaskApprovalWorkflow, který rozšiřuje *content* typ Task dostupný v systému SharePoint. Tento nový typ obsahuje nově vytvořené sloupce.
3. Nový *content* typ je potřeba propojit se seznamem WorkflowTasks, který existuje v systému SharePoint. Toho docílím deklarativním způsobem pomocí následující XML definice:

   ```
   <ContentTypeBinding ContentTypeId="0x010801003f4aec2d470e4a72b22276deeb9aafbd"
   ListUrl="WorkflowTasks" />
   ```

   V ukázce XML kódu je vidět GUID nově vytvořeného *content* typu a jakým způsobem je spojen se seznamem WorkflowTasks.

5.9.1.2. Implementace *workflow*

Na následujícím obrázku (Obrázek 21) uvádím náhled *workflow* z editoru Visual Studia, ze kterého bude jasněji vidět, jak je proces vytvořen pomocí WF frameworku 3.5. Na obrázku není *workflow* vyobrazeno celé, nicméně všechny podstatné aktivity jsou v obrázku zobrazeny. Dále uvádím popis osvětlující nejdůležitější aktivity a způsob jakým celé *workflow* funguje.



Obrázek 21 Vytvořené workflow v editoru Visual Studia 2010

Po spuštění *workflow* tedy vytvořím nový úkol *content* typu TaskApprovalWorkflow, který jsem pro tento účel vytvořil. Úkol je přiřazen uživateli, který byl vybrán ke schválení (revizi) dokumentu.Pro vytvoření nového úkolu je použita aktivita CreateTask, která je jednou ze standardních *workflow* aktivit pro systém SharePoint. V této aktivitě je definován GUID content typu, podle kterého bude úkol vytvořen.

Workflow dále pokračuje aktivitou Listen, která rozděluje běh do dvou větví. V každé větvi se čeká na vyvolání nějaké události. Provede se ta větev, ve které bude prvně vyvolána událost. V mém *workflow* v jedné větvi očekávám událost OnTaskChanged a v druhé vypršení aktivity Delay, ve které je specifikováno, kdy nejpozději se má dokument publikovat. U aktivity OnTaskChanged je samozřejmě uvedeno, ke kterému záznamu v seznamu úkolů se vztahuje. Za aktivitou Delay následuje aktivita Code. Tato aktivita spuští mnou vytvořený kód pro publikování dokumentu. Za aktivitou OnTaskChanged ještě následuje aktivita IfElse. Pokud je výsledek revize kladný, provede se opět přes Code aktivitu kód provádějící publikování.

Je každopádně dobré si povšimnout, že aktiva Listen musí být „obalena“ nekonečným While cyklem z důvodu, aby bylo neustále kontrolováno, zda neuplynula doba pro Delay aktivitu. Bez tohoto nekonečného cyklu není řešení funkční. Zjištění, že musí být tímto způsobem použita nekonečná While aktivita, mě zabralo mnoho času. Nekonečný cyklus ukončuji vlastním kódem v pozadí, jakmile dojde k vyvolání některé z událostí.

5.10. Implementace web parts

Web parts jsou základním stavebním blokem uživatelského rozhraní v systému SharePoint. *Web parts* mohou být přidány do stránek uživateli systému [1]. *Web parts* bývají použity především na aspx stránkách, které jsou uloženy v *content* databázi systému [2].

Ve svém intranetovém řešení jsem implementoval následující *web parts*:

BlogPostsTable

Web part, který zobrazuje agregované příspěvky z uživatelových oblíbených blogů.

WIKIPostsTable

Web part, který zobrazuje agregované příspěvky z uživatelových oblíbených WIKI

RememberFavoriteSite

Web part zobrazuje pouze jednoduchý odkaz, který slouží k uložení aktuální *web site* do oblíbených. *Web part* je automaticky vkládán na *web site* typu blog nebo WIKI.

FavoriteSites

Web part zobrazující uživatelem uložené oblíbené *web site* ve formě odkazů. Obsahuje také funkcionalitu pro přidání nové oblíbené *web site* a odstranění *web site* z oblíbených.

News

Web part zobrazující aktuální novinky ze všech *web site*, na které má uživatel přístupová práva.

MyProjects

Web part zobrazující všechny projektové *web site*, na kterých je uživatel členem. Předpokládá se, že ke každému projektu, na kterém uživatel pracuje, existuje nějaká projektová *web site*.

ProjectTasks

Slouží k zobrazení úkolů, které jsou uloženy v seznamech typu Task a přiřazeny konkrétnímu uživateli, ze všech projektových *web site*, u kterých je uživatel členem.

SubSitesView

Slouží k zobrazení *web site*, které v hierarchii leží pod aktuální *web site*. *Web part* je použit na kořenové *web site* pro blogy a WIKI. *Web part* je možno propojit s BlogPostsTable nebo WIKIPostsTable. *Web part* umožňuje přechod na některou *web site* nebo pouze její označení, které způsobí zobrazení nejnovějších příspěvků v BlogPostsTable.

StocksInformations

Web part zobrazující informace o aktuální hodnotě akcií načtené z veřejného zdroje (viz kapitola *5.8. Implementace logiky a uživatelského rozhraní pro načítání a zobrazení dat z veřejných zdrojů*).

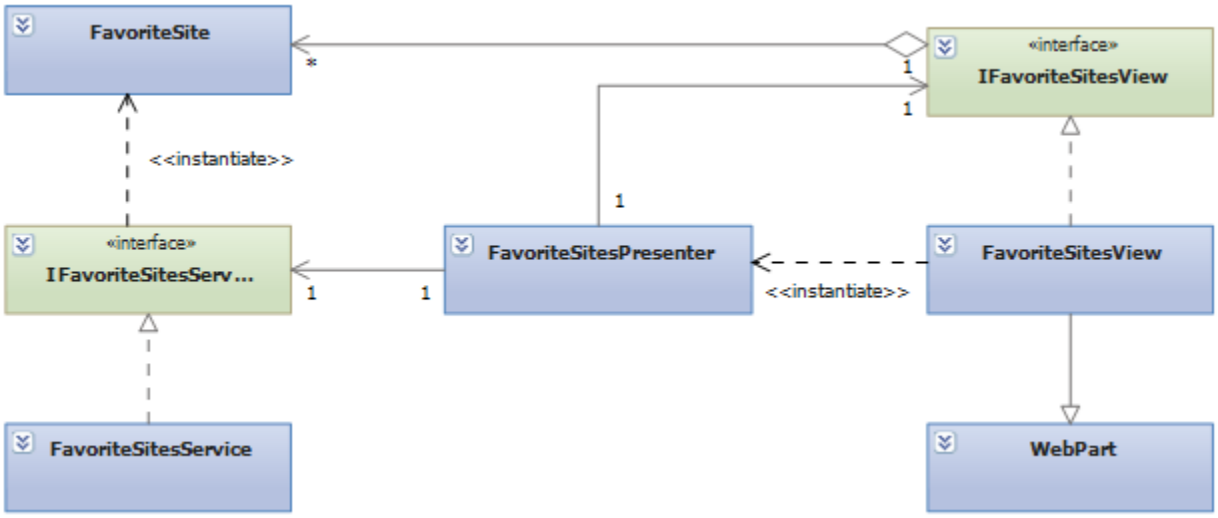
V následující kapitole nepopisuji implementaci všech zmíněných web parts, zaměřuji se na popis obecných pravidel použitých pro implementaci a na určité složitější web parts.

5.10.1. Návrhový vzor Model-View-Presenter

Existuje více přístupů jakým způsobem implementovat *web parts*. Já jsem se rozhodl pro implementaci zvolit návrhový vzor Model-View-Presenter. Tento přístup zajišťuje oddělení doménové logiky od implementace uživatelského rozhraní. Vzor je mimo jiné doporučovaný v knize *Designing Solutions for Microsoft SharePoint 2010* [6] pro implementaci ASP.NET komponent. Protože *web parts* jsou dalším rozšířením webových komponent ASP.NET považuji použití tohoto vzoru při implementaci *web parts* za velmi dobrou techniku.

Návrhový vzor rozděluje logiku mezi tři třídy. Model zapouzdřuje logiku přístupu k datům, Presenter získává data z modelu a předává je do View. Konečně View představuje samotné uživatelské rozhraní. Jedná se obvykle o pasivní třídy, které umožňují Presenter objektům vložit data, které později zobrazují [6].

Pro ukázku uvádím na následujícím class diagramu strukturu tříd FavoriteSites *web part* komponenty.



Obrázek 22 Class diagram pro WebPart FavoriteSites

Z class diagramu je vidět, že třída FavoriteSitesView dědí z třídy WebPart, což je bázová třída pro *web part* komponenty. Dále třída implementuje obecné rozhraní, které obsahuje referenci na kolekci FavoriteSite tříd, která představuje data určená k zobrazení. Tato kolekce je naplněna Presenter objektem (třída FavoriteSitesPresenter), který používá k načítání dat servisní třídu z doménové vrstvy. Presenter pracuje s obecným rozhraním IFavoriteSitesService, aby bylo možné konkrétní implementaci v případě potřeby nahradit.

5.10.2. Implementace connectable web parts

Knihovny systému SharePoint, sloužící k vytváření vlastních WebParts, obsahují množinu rozhraní nazvanou *connection interfaces*, která umožňují výměnu informací a zpráv mezi jednotlivými *web part* komponentami [1].

Connectable web parts jsem využil na kořenové *web site* pro blogy a WIKI, kde je použit SubSitesView w*eb part* a k němu připojen BlogsPostsTable respektive WIKIPostsTable. Tabulka zobrazená v SubSitesView umožňuje označení některého z blogů nebo WIKI. Na označení reaguje připojený WebPart a zobrazí několik naposledy vytvořených příspěvků. Automatické umístění a propojení obou WebPart při vytvoření kořenové *web site* blogů nebo WIKI již bylo popsáno dříve v kapitole *5.3.4. Site definition pro root web site blogů a WIKI*. BlogPostsTable a WIKIPostsTable jsem vytvořil primárně pro zobrazení agregovaných příspěvků. Nicméně pokud jsou propojeny se SubSitesView nezobrazují agregované příspěvky ale příspěvky z aktuálně vybrané *web site* ve SubSitesView. O tom, jaký typ příspěvků je načítán, je rozhoduto v *presenter* třídách BlogPostPresenter respektive WIKIPostsPresenter. Tyto třídy provedou korektní volání doménové vrsty pro načtení dat a předají k zobrazení do *view.*

Při implantaci *connectable web part* je jedna komponenta v roli poskytovatele a druhá v roli příjemce zpráv. Pro vytvoření poskytovatele a příjemce zpráv je nutné v rámci WebParts vytvořit metody, které

budou označeny atributy ConnectionProvider, respektive ConnectionConsumer. Obě metody pracují s určitým společným rozhraním. Metoda v poskytovateli používá rozhraní jako návratový typ, metoda v příjemci jako typ parametru [1].

Ukázka:

Ve svém řešení používám jako společné rozhraní IWebPartField, dostupné v knihovně systému SharePoint. V rámci poskytovatele je implementována metoda označená atributem ConnectionProvider.

```
[ConnectionProvider("List Of Blogs Provider", "SubSitesProvider")]
public Interface GetField()
{
    return this;
}
```

Příjemce obsahuje metodu označenou atributem ConnectionConsumer.

```
 [ConnectionConsumer("Blog URL Consumer", "BlogPostsConsumer", AllowsMultipleConnections
= false)]
public void SetFieldInterface(Interface field)
{
   var data = field.GetData();
   // Další zpracování
}
```

Návratový typ metody v poskytovateli dat se shoduje s typem parametru v příjemci dat. (*V ukázce není typ přesně specifikován*).

5.10.3. Web parts pro zobrazení novinek a úkolů

Web part News zobrazuje nejnovější aktuality ze všech publikačních *web site*, do kterých má uživatel přístup a *web part* ProjectTasks zase všechny úkoly, které jsou na různých projektových *web site* uživateli přiřazeny. Pro získání novinek a úkolů z více *web site* ale není v tomto případě použit agregační job a ukládání do dalšího seznamu, jednotlivé položky jsou načítány při každém načtení příslušné *web part*. Tento rozdílný přístup používám proto, že neočekávám příliš velké množství publikačních *web site* v rámci jedné *site collection*, respektive příliš velké množství projektů (a k nim projektových *web site*), na kterých by uživatel v jednom okamžiku pracoval. Načítání se také provádí ze všech publikačních a projektových *web site*, na nichž je uživatel členem nějaké skupiny s právy pro čtení. Neprovádí se podle uživatelem zvolených oblíbených *web site*, což celou situaci zjednodušuje.

Samotné načítání je řešeno pomocí třídy SPSiteDataQuery. Tato třída umožňuje získat data z určitého typu seznamu z celé *site collection* podle zadaných kritérií [1].

Ukázka načítání úkolů, které jsou přiřazeny určitému uživateli:

```
SPSiteDataQuery query = new SPSiteDataQuery();

query.Lists = "<Lists ServerTemplate=\"107\" />";

query.ViewFields = "<FieldRef Name=\"Title\" />" +
                   "<FieldRef Name=\"Created\" Nullable=\"TRUE\" Type=\"DateTime\"/>" +
                   "<FieldRef Name=\"CreatedBy\" Nullable=\"TRUE\" Type=\"User\"/>";

query.Query = "<Where>" +
                  "<Eq>" +
                     "<FieldRef     Name=\"AssignedTo\"/><Value     Type=\"Text\">"     +
              user.LoginName + "</Value>" +
                  "</Eq>" +
              "</Where>" +
              "<OrderBy>" +
                  "<FieldRef Name=\"Created\" Ascending='False' />" +
              "</OrderBy>";

query.RowLimit = 10;

query.Webs = "<Webs Scope=\"SiteCollection\" />";
DataTable dt = siteCollection.RootWeb.GetSiteData(query);
```

Vlastnost Lists objektu typu SPsiteDataQuery specifikuje typ seznamů (pomocí ID), ze kterých se mají získat data. ViewFields zase specifikuje, které sloupce ze seznamů mají být načteny. Query obsahuje samotný CAML dotaz, který je zapsán ve formátu XML. V elementu Where je podmínka pro načtení úkolů pouze pro specifikovaného uživatele. Element OrderBy pak slouží ke správnému seřazení výsledků. RowLimit určuje, že bude vráceno pouze 10 výsledků a konečně Webs specifikuje rozsah vyhledávání, v tomto případě *site collection*.

5.10.4. Asynchronní zpracování v rámci *web parts*

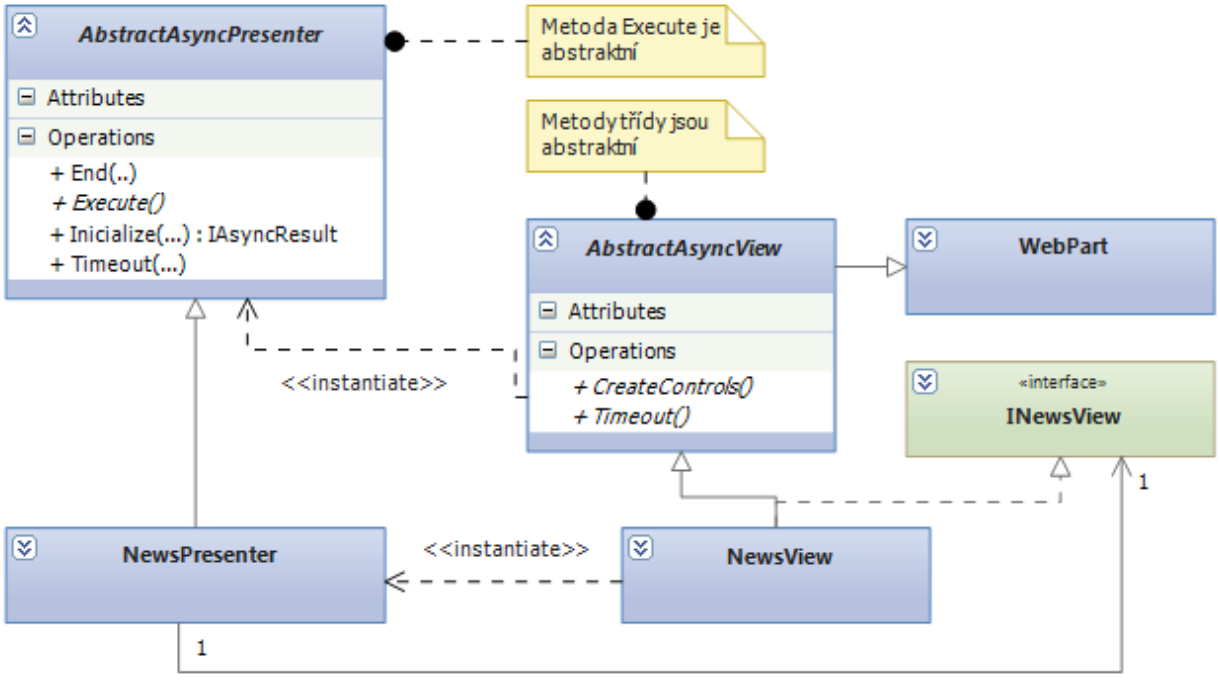
Načítání novinek nebo úkolů z celé *site collection* by mohlo zabrat příliš dlouhou dobu a tím zablokovat zobrazení celé stránky. Abych tomuto zamrznutí zabráznil, rozhodl jsem se tyto operace provádět asynchronně. Ve třídách, které dědí ze třídy WebPart, je možné provést zaregistrování asynchronní úlohy. Zaregistrování určitého asynchronního zpracování se provádí pomocí metody RegisterAsyncTask objektu Page. Metoda přijímá jako parametry několik delegátů, kteří představují metody, které se spustí při zahájení vykonávání asynchronního úkolu, při ukončení a případně při vypršení časového limitu.

Ukázka zaregistrování asynchronní úlohy v rámci *web part*:

```
var task = new PageAsyncTask(
                    new BeginEventHandler(_presenter.Initialize),
                    new EndEventHandler(_presenter.End),
                    new EndEventHandler(_presenter.Timeout),
                    null,
                    true);

Page.AsyncTimeout = TimeSpan.FromSeconds(5);
Page.RegisterAsyncTask(task);
```

V ukázce kódu je vidět nejprve vytvoření objektu typu PageAsyncTask a předání potřebných delegátů na metody. Nad objektem Page je dále nastaven limit, po jehož vypršení se asynchronní zpracování ukončí, a nakonec zaregistrování asynchronního úkolu metodou RegisterAsyncTask.

Protože asynchronní zpracování používám ve více *web part* (načítání novinek, úkolů, zobrazení týmových *web site,* kterých je uživatel členem), vytvořil jsem několik několik bázových tříd pro usnadnění implementace dalších *web parts.* Jsou zobrazeny na následujícím class diagramu v kontextu *web part* zobrazujícího novinky z celé *site collection*:



Obrázek 23 Implementace asynchronních web parts

Třída AbstractAsyncView zaregistruje metodu Inicialize (definovaná ve třídě AbstractAsyncPresenter) pro asynchronní zpracování způsobem ukázaným výše (pomocí objektu PageAsyncTask). Metoda End a

metoda Timeout představují metody, které spustí pokud je úloha dokončena respektive v případě neúspěšného dokončení úlohy ve stanoveném čase. Metoda Initialize vytvoří nový objekt typu Action a asynchronně spustí vykonávání metody Execute, viz následující ukázka kódu:

```
_task = new Action(Execute);
return _task.BeginInvoke(cb, extraData);
```

Metoda Execute je v bázové třídě abstraktní a musí být implementována v konkrétních třídách, které dědí z AbstractAsyncPresenter. V této metodě má být provedena operace, která by mohla trvat delší dobu a při sekvevenčním spuštění způsobit zamrznutí. V této metodě tedy budou volání doménové vrstvy za účelem získání dat. V případě úspěšného ukončení úlohy je zavolána metoda End, která dale volá metodu CreateControls třídy *view*. V případě, že úloha není v časovém limitu dokončena, je v *presenter* objektu vyvolána metoda Timetou, která dale volá stejnojmenou metodu ve *view*. Metody CreateControls a Timeout definované ve třídě AbstractAsyncView jsou abstraktní a musejí být implementovány v konkrétních *web parts*. Jejím účelem je vykreslení korektního výstupu uživateli v případě úspěšného respektive neúspěšného dokončení úlohy.

(poznámka: V class diagramu (Obrázek 23) jsou pro přehlednost v metodách třídy AbstractAsyncPresenter zanedbány parametry.)

5.10.5. Nastavitelnost web part komponent

U jednotlivých *web part* je možno definovat vlastnosti, které je možno nastavit přes uživatelské rozhraní systému SharePoint. Tato nastavení SharePoint uloží do databáze. Ve svém řešení samozřejmě také využívám možnost definovat nastavitelné vlastnosti, protože by například určitě nebylo vhodné, aby počet novinek, které zobrazuje komponenta News byl definován neměnitelně.

Vlastnosti, které mají být nastavitelné je potřeba ve třídě reprezentující *web part* označit speciálními atributy, viz následující ukázka kódu:

```
[Personalizable(PersonalizationScope.User)]
[WebBrowsable(true)]
[Category("Miscellaneous")]
[Description("")]
[WebDisplayName("Count of shown posts:")]
public int PostsCount
{
	get { return _postsCount; }
	set { _postsCount = value; }
}
```

Atribut WebBrowsable zajistí, aby byla vlastnost viditelná v editačním panelu, atribut Personalizable určuje, kteří uživatelé mohou nastavení měnit. Parameter PersonalizationScope.User určuje, že si každý může tuto vlastnost nastavit sám [1]. Při editaci w*eb part* jsou jednotlivé měnitelné vlastnosti rozděleny do kategorií. Kategorie je specifikována atributem Category. Význam dalších atributů je jasný již z jejich názvu.

Závěr

Ve své diplomové práci jsem vytvořil řešení nad systémem SharePoint 2010, které může sloužit jako základ pro další implementace zákaznických projektů společnosti Tieto. V rámci tohoto řešení jsem popsal způsob, jakým může být navržena architektura řešení postavených na systému SharePoint. Důležitou součásti této architektury je její další možná rozšiřitelnost a možnost nahrazení konkrétních implementací tříd pomocí frameworku Unity. Datová vrstva je mapována na třídy jazyka C# pomocí nového přístupu LINQ to SharePoint, který by měl stále více nahrazovat dotazovací jazyk CAML. Do své práce jsem zahrnul typické často se opakující požadavky, jako je agregace dat, rozšíření publikační funkcionality a funkcionalit, které se týkají blogů a WIKI. Implementoval jsem také jednoduchou integraci s externím systémem a proces pro publikování pomocí Workflow Foundation frameworku. Tyto implementace a jejich popis mohou dobře posloužit jako výchozí bod dalším programátorům při řešení složitějších scénářů.

V budoucnu by bylo dobré v systému dále rozšířit funkcionality týkající se blogů a WIKI, například filtrováním podle klíčových slov, vyhledáváním anebo *workflow* procesem pro schválení požadavku na založení nového blogu nebo WIKI.

Použitá literatura

[1] Joerg Krause, Martin Döring, Christian Langhirt, Bernd Pehlke, Alexander Sterff. SharePoint 2010 as a Development Platform. [s.l.] : Apress, 2010. 1164 s. ISBN 978-1-4302-2706-9.

[2] MALIK, Sahil. Microsoft SharePoint 2010 : Building Solutions for SharePoint 2010. [s.l.] : Apress, 2010. 400 s. ISBN 978-1-4302-2865-3.

[3] DE LA TORRE, César, Unai ZORRILLA, Javier CALVARRO a Miguel Angel RAMOS. N-Layered Domain-Oriented Architecture Guide with .NET 4.0. 1. vyd. Krasis Press, 2011. ISBN 978-84-939036-1-9.

[4] KLINDT, Shane YOUNG a Steve CARAVAJAL. *Professional SharePoint 2010 Administration*. USA: Wiley Publishing, Inc, 2010. ISBN 978-0-470-53333-8.

[5] PYLES, James. MCTS: Microsoft SharePoint 2010 configuration study guide. Indianapolis, Ind.: Wiley Pub., c2011, 616 s. ISBN 9780470627013 (PBK.).

[6] BAGINSKI, Todd, Lee JASON, Robert BOGUE. Designing solutions for Microsoft SharePoint 2010: making the right architecture and implementation decisions. Redmond, Wash.: Microsoft Press, 2010. ISBN 07-356-5608-8.

Internetové zdroje

[7] Licensing Details. Microsoft SharePoint 2010 [online]. [cit. 2012-02-28]. Dostupné z: http://SharePoint.microsoft.com/en-us/buy/pages/licensing-details.aspx

[8] Hardware and software requirements (SharePoint Server 2010). Microsoft Technet [online]. 8.7.2010 [cit. 2012-02-28]. Dostupné z: http://technet.microsoft.com/en-us/library/cc262485.aspx

[9] Topologies for SharePoint Server 2010: Model. Microsoft Technet [online]. 3.1.2011 [cit. 2012-02-29]. Dostupné z: http://technet.microsoft.com/en-us/library/cc263044.aspx

[10] Technical diagrams (SharePoint Foundation 2010). Microsoft Technet [online]. 17.6.2010 [cit. 2012-02-29]. Dostupné z: http://technet.microsoft.com/en-us/library/ee806874.aspx

[11] Microsoft Unity 2.0. MSDN Library [online]. Duben 2010 [cit. 2012-02-29]. Dostupné z: http://msdn.microsoft.com/en-us/library/ff663144.aspx

[12] SPMetal. MSDN Library [online]. Březen 2010 [cit. 2012-02-29]. Dostupné z: http://msdn.microsoft.com/en-us/library/ee538255.aspx

[13] ListTemplate Element (List Template). MSDN Library [online]. Březen 2010 [cit. 2012-02-29]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms462947.aspx

[14] File Element (Module). MSDN Library [online]. Březen 2010 [cit. 2012-02-29]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms459213.aspx

[15] Upgrading an Existing Master Page to the SharePoint Foundation Master Page. MSDN Library [online]. Březen 2010 [cit. 2012-03-01]. Dostupné z: http://msdn.microsoft.com/en-us/library/ee539981.aspx

[16] Starter Master Pages for SharePoint [online]. 8.10.2010 [cit. 2012-03-01]. Dostupné z: http://startermasterpages.codeplex.com/

[17] Disposing Objects: SharePoint 2010. MSDN Library [online]. Březen 2010 [cit. 2012-03-31]. Dostupné z: http://msdn.microsoft.com/en-us/library/ee557362.aspx

[18] Programmatically Update Page Layouts. SharePointers [online]. 4.9.2008 [cit. 2012-04-13]. Dostupné z: http://sharepointers.blogspot.com/2008/09/programmatically-update-page-layouts.html